ON MATRICES THAT DO NOT HAVE THE CONSECUTIVE ONES PROPERTY

by

Vivija Ping You

B.Sc. University of Victoria, 2007

a Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in the Department
of
Mathematics

SIMON FRASER UNIVERSITY
Summer 2009

APPROVAL

Name: Vivija Ping You

Degree: Master of Science

Title of Thesis: On Matrices that Do Not Have the Consecutive Ones Property

Examining Committee: Dr. Tamon Stephen
Chair

Dr. Cedric Chauve, Associate Professor, Mathematics, Simon Fraser University
Senior Supervisor

Dr. Ladislav Stacho, Associate Professor, Mathematics, Simon Fraser University
Supervisor

Dr. Jan Manuch, Postdoctoral Fellow, Computing Science, Simon Fraser University
Supervisor

Dr. Marni Mishna, Assistant Professor, Mathematics, Simon Fraser University
SFU Examiner

Date Approved: _____

Abstract

A binary matrix has the consecutive ones property if its columns can be ordered in such a way that, in each row, all 1s are consecutive. This classical combinatorial notion has been central in genomic problems such as physical mapping or paleogenomics. In these fields, genomes that cannot be sequenced are represented by a matrix that has the consecutive ones property, but are inferred from an initial matrix that does not have this property due to errors. In this work, we study combinatorial and algorithmic characterizations of matrices that do not have the consecutive ones property. We review existing results and propose new results centered around the notion of minimal conflicting sets.

Contents

List of Tables

List of Figures

This thesis is dedicated to my mom, YunLan Yang, and to Venerable Guan Cheng of the International Buddhist Temple in Richmond, British Columbia.

Acknowledgments

I want to thank my senior supervisor Dr. Cedric Chauve, who introduced me to this interesting mathematics, and whose dedication and insight helped shape the thesis. I also thank the committee members Dr. Tamon Stephen, Dr. Ladislav Stacho, Dr. Jan Manuch, and Dr. Marni Mishna, and others who read my thesis and gave me advice: Steve Kieffer, Suling Yang and Sam Bassett. I thank Dr. Peter Dukes and Justin Chan at the University of Victoria, who encouraged me and were always supportive and patient, as well as my classmates at U-Vic: Steve Lowdon, Jian Kang, Philip Rempel. I also want to thank all the professors who wrote references for me, and finally all my coworkers in the calculus workshop, and the coordinators Justin Gray, and Keshav Mukunda.

Part I

Background

Chapter 1

Introduction

Binary matrices, i.e. matrices whose entries are either 0 or 1, are classical combinatorial objects that have been used in several types of applications. For a given binary matrix, if we can rearrange its columns such that the 1s in each row are consecutive, we say that the matrix has the consecutive ones property. The consecutive ones property plays a central role in several applications in which we want to arrange a set of objects such that some of the objects are required to be contiguous. In computational biology for example (the main motivation for the work presented in this thesis), we want to verify whether a given set of segments of chromosome is compatible with a linear arrangement of the genes it contains [33]. In graph theory, it has been used to test whether a given graph is an interval graph [22]. In file organization, in a computer file system for example, we want to arrange the records such that the response to each query can be retrieved as a set of consecutive records [19]. In statistical archaeology, we want to get the information from the 'graves-versus-varieties' matrix to see whether each variety of pottery can be ascribed to a definite segment of the true temporal order [28].

In this introductory chapter, we first recall the formal definition of the consecutive ones property, together with some related graph theoretical notions and a brief historical survey. Next, we will present two important applications of the consecutive ones property in computational biology: physical mapping and paleogenomics. This discussion will lead naturally to the main objects we study in this thesis: matrices that do not have the consecutive ones property, and minimal conflicting sets. We will then present some existing results for the case of matrices with exactly two 1s per row. These results suggest research questions for the general case, that we attack in the current thesis. We will conclude this chapter by

outlining the plan of the rest of this thesis and the results we obtained.

1.1 The Consecutive Ones Property

1.1.1 Formal Definitions

We give here the formal definition of the consecutive ones property.

Definition 1.1.1 Let m and n be two positive integers. An $m \times n$ *binary matrix* M is a matrix with m rows and n columns, with entries equal to 0 or 1. We denote by r_i the i^{th} row and c_j the j^{th} column of M, and by M_{ij} the entry on the i^{th} row and j^{th} column. We denote by $R(M)$ (resp. $C(M)$) the set of rows (resp. columns) of M, and we write c for the total number of 1s in M.

Definition 1.1.2 A binary matrix has the *consecutive ones property* ($C1P$ from now on) if there exists a permutation of its columns such that after ordering the columns according to this permutation, all the entries equal to 1 in each row are consecutive. Binary matrices that have the $C1P$ are called $C1P$ matrices, and binary matrices that do not have the $C1P$ are called non-$C1P$ matrices.

Remark 1.1.3 Without loss of generality, we assume for the rest of this paper that the binary matrices we are considering contain no identical rows or columns, the number of $1s$ in each row is at least 2, and the number of $1s$ in each column is at least 1.

Definition 1.1.4 Let M be a binary matrix. A permutation of $C(M)$ is *valid for* M if after ordering the elements of $C(M)$ according to this permutation, the $1s$ in each row are consecutive. We say the permutation is *invalid for* M otherwise.

We introduce now a natural set-theoretical alternative way of viewing the $C1P$ for a given binary matrix.

Definition 1.1.5 For a given binary $m \times n$ matrix M, we represent the i^{th} row r_i by the set $\{1 \leq j \leq n : M_{ij} = 1\}$, also denoted by r_i. We then use $R(M)$ to also denote the set $\{r_1, \ldots, r_m\}$ of subsets of $\{1, \ldots, n\}$.

The following property follows immediately from this definition and the definition of valid permutations.

Definition 1.1.6 If π is a permutation, then an *interval* of π is the image $\pi([a, b])$ of any interval $[a, b] \subseteq \{1, \ldots, n\}$.

Proposition 1.1.7 *A permutation π of $C(M)$ is* valid *for M if and only if, for every set $r_i = \{j_1, \ldots, j_k\}$ of $R(M)$, the subset $\{c_{j_1}, \ldots, c_{j_k}\}$ of $C(M)$ is an interval of π.*

Example 1.1.8 Consider the following binary matrix:

	c_1	c_2	c_3	c_4
r_1	**1**	**1**	0	0
r_2	0	**1**	**1**	**1**
r_3	0	**1**	0	**1**

Its set-theoretic representation is $R(M) = \{\{1, 2\}, \{2, 3, 4\}, \{2, 4\}\}$.
If we order the columns as c_1 c_2 c_4 c_3, we have:

	c_1	c_2	c_4	c_3
r_1	**1**	**1**	0	0
r_2	0	**1**	**1**	**1**
r_3	0	**1**	**1**	0

As we can see that the $1s$ in each row are consecutive after we rearrange the columns, therefore this is a $C1P$ matrix.

Conversely, the following binary matrix is a non-$C1P$ matrix.

	c_1	c_2	c_3	c_4
r_1	**1**	**1**	0	0
r_2	0	**1**	**1**	0
r_3	0	**1**	0	**1**

From row r_1, $\{c_1, c_2\}$ should be an interval of any valid permutation. Similarly, from r_2 (resp. r_3), $\{c_2, c_3\}$ (resp. $\{c_2, c_4\}$) should be an interval in any valid permutation. This implies that c_2 should be adjacent to c_1, c_3 and c_4 in any valid permutation, which is impossible.

1.1.2 Two Graphs Related to the Consecutive Ones Property

We present here two graphs that are related to the $C1P$. The first one, a *bipartite graph*, is another representation of a binary matrix, while the second one, an *overlap graph* allows us to consider the problem of deciding if a matrix is a $C1P$ matrix as a set of independent simpler problems.

Definition 1.1.9 For a given binary matrix M, we define the bipartite graph associated to M as $B(M) = (V_1, V_2, E)$ where $V_1 \cup V_2$ is the set of vertices, and E is the set of edges. V_1 is the set of columns of M and V_2 is the set of rows of M and E is the symmetric adjacency relation defined on $V_1 \times V_2$ such that $(c_i, r_j) \in V_1 \times V_2$ is an edge if and only if $M_{ij} = 1$.

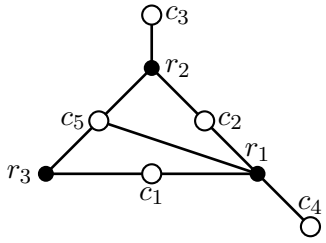
Remark 1.1.10 We call each vertex in V_1 a column vertex and each vertex in V_2 a row vertex in $B(M)$.

We can then translate the $C1P$ for the rows of M into an equivalent condition on the associated bipartite graph $B(M)$; namely, that the vertices of V_1 can be ordered such that for each $r_j \in V_2$, the set $N(r_j) = \{c_i \in V_1 : (c_i, r_j) \in E(G)\}$ appears to be consecutive in V_1, possibly with $N(r_j) = \emptyset$. Such an ordering is called a V_1-*consecutive arrangement* of $B(M)$.

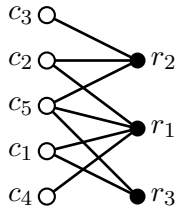
Example 1.1.11 For example, if we have the following matrix M:

	c_1	c_2	c_3	c_4	c_5
r_1	**1**	**1**	0	**1**	**1**
r_2	0	**1**	**1**	0	**1**
r_3	**1**	0	0	0	**1**

the corresponding bipartite graph $B(M)$ would be:



The permutation putting the columns in the order c_3, c_2, c_5, c_1, c_4 is valid for this matrix, and the corresponding V_1-consecutive arrangement appears as:



We now introduce the overlap graph of a binary matrix.

Definition 1.1.12 For a binary matrix M with the set of rows $\{r_1, r_2, \ldots, r_n\}$, the *overlap graph* $O(M) = (V, E)$ corresponding to M is defined by $V = \{r_1, r_2, \ldots, r_n\}$ and $E = \{(r_i, r_j) : r_i \cap r_j \neq \emptyset, r_i \not\subseteq r_j, r_j \not\subseteq r_i\}$.

It is then fairly straightforward to show the following important result:

Proposition 1.1.13 *A binary matrix M has the $C1P$ if and only if, for every component C of $O(M)$, its vertex set $\{r_{i_1}, \ldots, r_{i_k}\}$ has the $C1P$.*

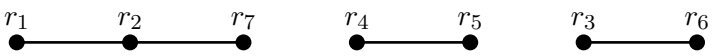
Proof. ($\Rightarrow$): Suppose for some component C, its vertex set $\{r_{i_1}, \ldots, r_{i_k}\}$ does not have the $C1P$. Then $R(M)$ does not have the $C1P$ since $\{r_{i_1}, \ldots, r_{i_k}\} \subseteq R(M)$.

($\Leftarrow$): Now suppose the vertex set of every component of $O(M)$ has the $C1P$. If $O(M)$ only has one component, then obviously M has the $C1P$. If $O(M)$ has more than one component, then, by the definition of $O(M)$, for any two components, either their vertex sets are disjoint, or one vertex set is contained in the other. Thus, combining valid permutations for each component will give a valid permutation for M. □

Example 1.1.14 Let M be the binary matrix illustrated below:

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}
r_1	**1**	0	0	0	0	0	**1**	0	0	**1**
r_2	0	0	**1**	0	0	**1**	**1**	0	**1**	**1**
r_3	0	0	0	**1**	**1**	0	0	0	0	0
r_4	0	0	**1**	0	0	**1**	0	0	0	0
r_5	0	0	0	0	0	**1**	0	0	**1**	0
r_6	0	**1**	0	**1**	0	0	0	0	0	0
r_7	0	0	**1**	0	0	**1**	0	**1**	**1**	0

Then $r_1 = \{1, 7, 10\}, r_2 = \{3, 6, 7, 9, 10\}, r_3 = \{4, 5\}, r_4 = \{3, 6\}, r_5 = \{6, 9\}, r_6 = \{2, 4\}, r_7 = \{3, 6, 8, 9\}$ and the corresponding overlap graph is:



$O(M)$ has three components: $C_1 = \{r_1, r_2, r_7\}$, $C_2 = \{r_4, r_5\}$ and $C_3 = \{r_4, r_5\}$. C_1 has the $C1P$, and it is easy to see that in every valid permutation restricted to C_1, such as

1 7 10 3 6 9 8 for example, the elements 3, 6 and 9 form an interval but their order does not matter. Therefore, since $r_4 \cup r_5 = \{3, 6, 9\}$, any permutation π valid for C_2 can be defined as necessary on {1 7 8 10} to be valid for C_1 as well. Since $r_3 \cup r_6 = \{2, 4, 5\}$ is disjoint from the union of the vertex sets of C_1 and C_2, we can further extend π to be valid for C_3 on {2 4 5}. A valid permutation for M would be: 1 7 10 3 6 9 8 2 4 5.

1.1.3 A Brief Historical Survey

The Consecutive Ones Property was first introduced by Fulkerson and Gross [17] in 1965 to determine whether the blemished portions of each pair of a given set of mutant genes intersect or not. They gave a polynomial time algorithm which can decide whether a binary matrix has the $C1P$, and, if so, constructs a valid permutation. Since then, the property has been considered in many areas, such as graph theory, theoretical computer science, and computational biology.

In 1972, Tucker [41] used the concept of an *asteroidal triple* to characterize the $C1P$ matrices in terms of five "forbidden" substructures. However, although Tucker's results provide a characterization of $C1P$ matrices, they do not translate into an efficient algorithm to decide if a matrix has the $C1P$.

It was only in 1975 that such an algorithm was proposed for the first time. This algorithm is due to Booth and Lueker [4] and uses the notion of *PQ-trees*. The PQ-tree associated to a $C1P$ matrix encodes all valid permutations using space that is polynomial in the size of the matrix. This data structure has since then been used in several other applications to encode sets of permutations.

Still, the initial algorithm by Booth and Lueker to compute a PQ-tree was quite complicated to implement, and it left the question of a certificate for non-$C1P$ matrices open. This was resolved in 2004 by McConnell [31], who, following some earlier work on partition refinements [22], introduced the notion of *PQR-tree* (a generalization of PQ-trees), and of *incompatibility graph*. (Briefly, a matrix has the $C1P$ if its PQR-tree has no R-node, making it in fact a PQ-tree. In that case, this PQ-tree encodes all the valid permutations for the matrix. Meanwhile, an odd length cycle in the incompatibility graph serves as a non-$C1P$ certificate.)

All the notions we briefly introduced here are discussed in greater detail in Chapter 2.

1.2 Motivation: Applications from Computational Genomics

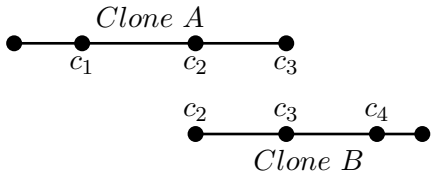
We present now two applications of the $C1P$ in computational biology that motivated the work presented in this thesis: physical mapping, and ancestral genome reconstruction. We refer the reader to [33] for an introduction to computational biology.

1.2.1 Physical Mapping

A *physical map of a segment of DNA* describes the location of certain markers along that DNA molecule, called from now on the *target DNA*. The markers are typically small but precisely defined DNA sequences, and are called *probes*. Physical maps were used in the pre-sequencing era, when sequencing a genome was impossible, to give a sequence of probes representing a rough description of the target DNA, less precise than the complete DNA sequence. The set of probes (which can contain several hundred different ones) can then be seen as a higher order alphabet than the classical $\{A, C, G, T\}$ nucleotide alphabet used to describe a genome at the finest scale.

Physical maps can be obtained experimentally using a technique called *hybridization*. The first step consists in obtaining several copies of the target DNA, and breaking each copy, randomly, into fragments called *clones*. For a given set of probes and a given clone C, the *fingerprint* of C is the (unordered) set of probes that appear as subsequences of C (remember that each probe is a short DNA sequence and C is a large unsequenced DNA segment). In order to determine experimentally the fingerprint of a clone C, each probe is dyed with a different color, and all are allowed to sit in a test tube with the clone C. Then, probes that occur in C will bind to C, and an image analysis of the clone after the hybridization experiment will allow its fingerprint to be determined. In order to create a physical map from the set of the fingerprints of all clones (up to a few thousand clones can be considered), the idea is that if two clones share part of their respective fingerprints, then they are most likely from overlapping regions of the target DNA. If all overlaps between all clones are determined, and if all of the fingerprints are correct, then the clones can be ordered, yielding the order of the probes along the target DNA.

Example 1.2.1 In the figure, probes c_1, c_2, c_3 are bound to clone A, and probes c_2, c_3, c_4 to clone B. Hence it is reasonable to suppose that clone A and clone B overlap. From this we gain some partial information about the order of the probes along the target DNA.



Now if we construct a clones×probes binary matrix M where we set entry M_{ij}=1 if probe j binds to clone i and 0 elsewhere, then obtaining a physical map from M is equivalent to finding a valid permutation for this clones×probes binary matrix, and we should always be able to find such a permutation presuming no errors occurred during the hybridization process. Note that in fact the approach we just described assumes that each probe appears exactly once in the target DNA. The more general case where probes can bind in more than one location of the target DNA is more complicated.

1.2.2 Ancestral Genome Architecture

As seen above, physical maps were used to represent genomes when genome sequencing was not possible due to technological limitations. Currently physical maps are less used due to the availability of efficient sequencing techniques, and sequencing the genome of living species, especially higher order animals, is almost routine. However, all current species evolved from ancient organisms, that have disappeared. For example, it is believed that all placental mammalians evolved from a single species that was living approximately 125 million years ago [26]. From an evolutionary biology point of view, knowing the genome sequence of such ancestral species would be invaluable in order to understand the forces driving evolution; this recent field of research is called *paleogenomics* and aims at reconstructing ancestral genomic characters.

However, as a molecule, DNA degrades rapidly after a few hundred thousand years. So, even if fossils can be available for such ancestral species [26], it is impossible to extract quality DNA from these fossils. The only hope to obtain information on ancestral genomes relies on computational techniques, aimed at inferring the genome of ancient species from the genome of current species. This general problem received a lot of attention during the last few years, especially in the case of mammalian genomes, due to its implications for human evolution (see [10] and references therein).

We describe now a general approach for ancestral genome reconstruction, that re-uses the general principle of physical mapping and is centered on the consecutive ones property [10]. In this approach, the analog of the alphabet of probes is a set of *genomic markers* (that can

be genes or long DNA sequences) that are believed to have been present once and only once in the ancestral genome. An *ancestral synteny* is a set of such genomic markers that are believed to have been contiguous in the ancestral genome. Ancestral syntenies play a role analogous to the fingerprints in physical mapping, and a set of ancestral syntenies can then be encoded by a binary matrix, whose columns are the genomic markers and rows are the ancestral syntenies. If this matrix has the $C1P$, then every valid permutation represents a possible genome architecture for the ancestral genomes. The condition of consecutivity of the 1s in the rows of the matrix follows from the hypothesis that ancestral syntenies contain markers that were contiguous in the ancestor. Ancestral syntenies are obtained by comparing the genomes of current organisms: the general principle described in [10] is that if a set of genomic markers is contiguous in the genomes of two species whose evolutionary path goes through the sought ancestor, then it makes sense to assume these markers define an ancestral synteny.

This general approach has been used in several papers (see [10] and references there) and has led to a relative consensus on the karyotype (number of chromosomes) and general architecture of genome of the ancestor of placental mammalians. More distant organisms are however still challenging [6].

1.2.3 Real Data and $Non - C1P$ Matrices

From the previous two sections, it is clear that in computational biology, if no error occurs when generating a binary matrix, this matrix has the $C1P$. However in most of the cases, errors will occur, thus we will not get a $C1P$ matrix. More precisely, in both applications we discussed we can expect that most rows of the considered binary matrix represent correct information (i.e. correct fingerprints for some genome segments, or sets of markers that were contiguous in the ancestral genome) while some contain errors. The subset of rows that represent correct information, by the fact it encodes segments of an existing genome, defines a matrix that has the $C1P$, while the whole matrix does not have the $C1P$ due to the incorrect rows. The major difficulty in applications is then to detect the incorrect rows in order to discard them and obtain a matrix that has the $C1P$: this problem of detecting incorrect rows in paleogenomics applications is the main motivation for our work, and up to now has not been studied.

Detecting such incorrect rows is a hard problem, as they have been obtained through the same experimental process as correct rows (hybridization in physical mapping or comparison

of existing genomes in paleogenomics). The natural approach would then be to improve the methods used to obtain a binary matrix. However, this approach, which is more of a bioinformatics problem than a combinatorial problem, has not been successful up to now. Hence, most approaches are combinatorial in nature and consider only the binary matrix that does not have the $C1P$, and try to transform it into a matrix that has the $C1P$ via some combinatorial modifications. We discuss now this approach, and we concentrate on physical mapping, as it has a longer history than paleogenomics. We first describe typical errors in physical mapping and then we discuss some methods that transform a non-$C1P$ matrix into a $C1P$ matrix.

When we try to compute a physical map of an unknown genome, there are many possible errors that can happen during the hybridization experiments. For example:

1. A probe may fail to bind where it should which results in a false negative.

2. A probe may bind where it should not which results in a false positive.

3. During the cloning process, two pieces of target DNA may join and be replicated as if they were one clone. The resulting clone is called a chimeric clone, and in fact $40\% - 60\%$ of clones are chimeric. [33]

If errors do occur, it is very likely that the resulting binary matrix does not have the $C1P$.

Here we introduce a way to handle errors in hybridization experiments. We first examine the relationship between errors and the *gaps* in a clones$\times$probes binary matrix M. For every row r in M, if no errors are present, then all 1s in r should be consecutive. If r is a chimeric clone, then we should see some 0s (a gap) separating a block of 1s, resulting in two blocks of 1s. If r has a false positive somewhere, then it may separate a block of 0s into two by a 1, thus causing another gap. If r contains a false negative somewhere, then there may be a block of 1s split by a 0, and hence another gap may arise. As we can see that there is a correspondence between errors and gaps in M, therefore it is reasonable for us to think that if there is a valid permutation for M, then that permutation should have a minimum number of gaps. Thus, *gap minimization*, which asks for such an optimal permutation, can be considered as a generalization of the consecutive ones problem where a non-$C1P$ matrix is transformed into a $C1P$ matrix by minimizing the number of errors.

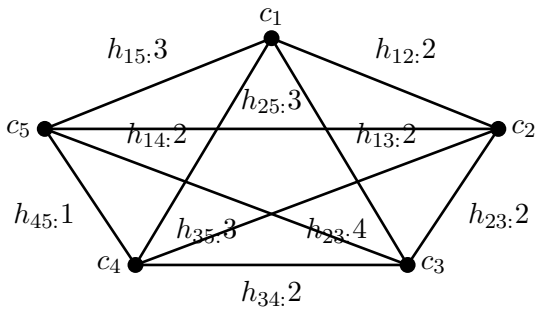
It turns out that we can reduce the gap minimization problem (GMP) to the traveling salesman problem (TSP) in which we want to find a minimum weighted Hamiltonian cycle

in a complete edge-weighted graph. [33] We transform M into a complete edge-weighted graph G in the following way: We attach one column consisting of all 0s to the last column of M, and call the new matrix M again; each column in M corresponds to a vertex in G, and the Hamming distance between two columns is the weight of the edge joining the two corresponding vertices.

Example 1.2.2 For example, we have a 4×4 binary matrix M with c_5 appended as illustrated below:

	c_1	c_2	c_3	c_4	c_5
r_1	**1**	**1**	0	0	0
r_2	0	**1**	**1**	0	0
r_3	**1**	0	**1**	**1**	0
r_4	**1**	**1**	**1**	0	0

If h_{ij} denotes the Hamming distance between columns c_i and c_j, then the corresponding graph G would be:



Theorem 1.2.3 *[33] A minimum-weighted Hamiltonian cycle in G induces a permutation of the columns of M with least number of gaps.*

It follows immediately from the link between the gap minimization problem and the Hamiltonian path problem that solving the gap minimization problem is NP-complete. In fact, even simpler decision problems, such as deciding if there exists a permutation of the columns of M such that in each row the number and length of gaps is bounded, are also NP-complete [9].

There are other approaches to transforming a non-$C1P$ matrix to a $C1P$ matrix, using combinatorial criteria. The most naive ways are to remove rows or columns [38, 18], with the criterion that the number of rows or columns removed be at a minimum, or to remove both

rows and columns, and try to maximize the size of the resulting matrix [12]. The rationale for removing rows is to remove fingerprints that are wrong, and removing columns corresponds to discarding probes that have been poorly selected and lead to errors. Swapping some 0s to 1s and some 1s to 0s by a minimum number of swaps [3] has also been considered as a way to correct hybridization errors. However, not surprisingly, all of the above problems have been proved to be NP-hard.

1.3 Conflicting Sets

We introduce in this section the main object of our study: minimal conflicting sets. After the definition of this concept, we discuss the case of matrices with two 1s per row.

1.3.1 Definitions

Definition 1.3.1 A subset of rows $R = \{r_{i_1}, r_{i_2}, ..., r_{i_k}\}$ of a binary matrix M is a *conflicting set* if the matrix defined by R does not have the $C1P$. R is a *minimal conflicting set* if R is a conflicting set but no proper subset of R is a conflicting set. We abbreviate minimal conflicting set(s) as MCS.

Definition 1.3.2 A row r of a binary matrix M is a *conflicting row* if it belongs to at least one minimal conflicting set. The *conflicting index* (CI) of r is the number of minimal conflicting sets it belongs to.

Example 1.3.3 $R = \{\{1,2\},\{2,3\},\{1,3\}\}$, which corresponds to the following matrix, forms a minimal conflicting set as one can check that R does not have the $C1P$, but deleting any row from R leaves two rows, which clearly have the $C1P$. This also shows that each of the $r_i \in R$ is a conflicting row.

	c_1	c_2	c_3
r_1	**1**	**1**	0
r_2	0	**1**	**1**
r_3	**1**	0	**1**

Example 1.3.4 For the matrix given below, one can verify that $R_1 = \{r_1, r_2, r_3\}$ and $R_2 = \{r_1, r_2, r_4\}$ are the only two minimal conflicting sets. Therefore the conflicting indices for r_1, r_2, r_3, r_4 are 2, 2, 1, 1, respectively.

	c_1	c_2	c_3	c_4
r_1	**1**	**1**	0	0
r_2	0	**1**	**1**	0
r_3	0	**1**	0	**1**
r_4	**1**	0	**1**	0

The notion of minimal conflicting set appears natural when one faces non-$C1P$ matrices as only such matrices contain minimal conflicting sets, and these structures are the minimal ones that cause a matrix to be non-$C1P$. Hence, for computational biology applications, errors in experiments (hybridization or detection of ancestral syntenies) that lead to non-$C1P$ matrices also result in the creation of minimal conflicting sets. The motivation of this thesis is to understand better the combinatorial structure of minimal conflicting sets in order to use this concept to detect errors in non-$C1P$ matrices obtained from genomic data. In particular, we are interested in two questions:

1. Given a row r of a non-$C1P$ matrix M, is $CI(r) > 0$?
2. Given a row r of a non-$C1P$ matrix M, what is $CI(r)$?

The second and more general question is hard, as we show in the next section, but has been asked in [37] as a piece of a branch-and-bound algorithm. The first question is important as one can expect that a row r of a non-$C1P$ matrix that does not belong to any minimal conflicting set need not be considered when trying to detect rows that contain errors. As most methods to detect error rows rely on computationally expensive algorithms, identifying rows that need not be considered can significantly increase computation speed.

1.3.2 Matrices with Two 1s per Row

We study here the simplest case, i.e. matrices with exactly two 1s per row. Such matrices, besides being simple, have been important in ancestral genome reconstruction, as their rows describe *adjacencies* (pairs of consecutive markers), a kind of syntenic information often used in genomics. We show that minimal conflicting sets have a simple structure in such matrices, but that computing the CI of the rows is a hard problem.

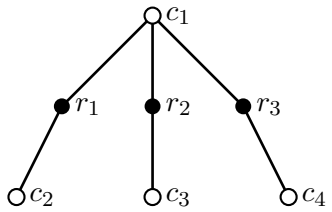
Definition 1.3.5 A *claw* in a binary matrix M with two 1s per row is a set $\{r, r', r''\}$ of rows such that there exists $i \in r \cap r' \cap r''$ and $j \in r \backslash (r' \cup r'')$, $j' \in r' \backslash (r \cup r'')$ and $j'' \in r'' \backslash (r \cup r')$.

We call i the root of a claw in $B(M)$.

Example 1.3.6 The following matrix M is a claw:

	c_1	c_2	c_3	c_4
r_1	**1**	**1**	0	0
r_2	**1**	0	**1**	0
r_3	**1**	0	0	**1**

The name claw comes from the structure of the corresponding bipartite graph $B(M)$ which is illustrated below:



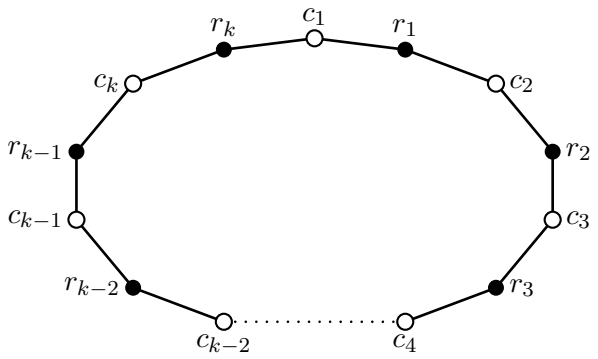
Remark 1.3.7 We see that the root of a claw is a column vertex. This root has three neighbors, each of which is adjacent to a different column vertex. If we try to get a valid permutation for the matrix corresponding to a claw, we see that three columns are forced to be "next to" the root column. This is impossible, so we cannot find a valid permutation for this matrix.

Definition 1.3.8 A *cycle* in a binary matrix M with two 1s per row is a set $\{r_{i_1}, r_{i_2}, \ldots, r_{i_k}\}$, where $k \geq 3$, of rows such that for every $j = 2, \ldots, k$, $|r_{i_{j-1}} \cap r_{i_j}| = 1$ and $|r_{i_1} \cap r_{i_k}| = 1$.

Example 1.3.9 For example, the following matrix M, where $k \geq 3$, is a cycle:

	c_1	c_2	c_3	c_4	$\ldots$	c_{k-2}	c_{k-1}	c_k
r_1	**1**	**1**	0	0	$\ldots$	0	0	0
r_2	0	**1**	**1**	0	$\ldots$	0	0	0
r_3	0	0	**1**	**1**	$\ldots$	0	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
r_{k-2}	0	0	0	0	$\ldots$	**1**	**1**	0
r_{k-1}	0	0	0	0	$\ldots$	0	**1**	**1**
r_k	**1**	0	0	0	$\ldots$	0	0	**1**

The corresponding structure in $B(M)$ is a cycle as illustrated below:



Remark 1.3.10 It is clear that, if a binary matrix M with two 1s per row is a $C1P$ matrix, then the graph $B(M)$ is a collection of disjoint paths. It then follows that cycles and claws, which are the minimal structures in a graph that are not paths, correspond to minimal conflicting sets. This gives a precise combinatorial characterization of minimal conflicting sets for such matrices.

Theorem 1.3.11 (folklore) *A binary matrix M with two 1s per row is a minimal conflicting set if and only if $B(M)$ is either a claw or a cycle.*

This characterization of conflicting sets lead to an answer to the two questions that we are interested in. We show below that deciding if a row belongs to at least one minimal conflicting set can be done in polynomial time, while computing the conflicting index is hard.

Corollary 1.3.12 *Given a row r of a binary matrix M with two 1s per row, deciding if $CI(r) > 0$ can be done in polynomial time.*

Proof. It follows from Theorem 1.3.11 that to decide if $CI(r) > 0$ for a given row r, it is sufficient to check whether r belongs to a claw or a cycle. To decide whether r belongs to a claw, we only need to consider all triples of rows that contain r, and there are $O(n^3)$ such triples. To decide whether r belongs to a cycle, we need to check whether there is a path from r_x to r_y in $B(M)$ that avoids r, where r_x, r_y are two row vertices in $B(M)$ such that $r_x \cap r \neq \emptyset \neq r_y \cap r$. This can be answered in time $O(n^2m^2(m+n)^2)$. □

For the next result, we will use the notion of #P-hard problem, defined by Valiant (see [42] for example) for counting problems. The #P problems are the counting problems corresponding to the decision problems in NP. #P-complete and #P-hard are defined analogously to NP-complete and NP-hard.

Corollary 1.3.13 *The problem of computing the conflicting index for a given row in a binary matrix with two* 1*s per row is #P-hard.*

Proof. To prove this hardness result, we reduce the classical #P-hard problem of counting the number of paths between two vertices of a graph to computing the number of cycles containing a given edge. Given a graph G and two vertices s and t in G, Valiant showed in [42] that computing the number of paths between s and t is #P-hard. It follows that, given an edge $\{s,t\}$ in a graph G, counting the number of cycles in G that contain this edge is #P-hard, as it is equivalent to counting the number of paths between s and t in $G-\{s,t\}$.

Let G be a graph, $\{v_1,\ldots,v_n\}$ be the vertices of G and $\{e_1,\ldots,e_m\}$ the edges of G. We can build a matrix M_G from G as follows: M_G has n columns $\{c_1,\ldots,c_n\}$ and m rows $\{r_1,\ldots,r_m\}$ defined by $r_i = \{c_{j_1}, c_{j_2}\}$ if and only if $e_i = \{v_{j_1}, v_{j_2}\}$. This reduction can obviously be performed in polynomial time and space in the size of G.

Now consider a pair of vertices v_{j_1} and v_{j_2} of G such that $e_i = \{v_{j_1}, v_{j_2}\}$. Counting the number of paths between these two vertices is equivalent to counting the number of cycles containing r_i, as $B(M_G)$ is the graph G where a vertex has been added on each edge to make it bipartite. This reduction proves the corollary since we need to count the number of cycles containing r_i in order to compute $CI(r_i)$. □

Note however that the problem of listing all the cycles of a graph, although it can require an exponential time as there can be an exponential number of cycles, can be solved in time that is polynomial in the output, using a backtracking algorithm (see Section 4.3).

1.4 Results and Plan

The rest of this thesis deals with non-$C1P$ matrices and minimal conflicting sets.

In Chapter 2, we will describe the main approaches to decide if a binary matrix has the $C1P$ or not. We will describe in detail the Tucker patterns, partition refinement, PQ-trees, and PQR-trees.

In Chapter 3, we will first consider the case of matrices with exactly three 1s per row, and we will characterize the minimal conflicting sets for such matrices by proving a generalization of Theorem 1.3.11. Although this is the next simplest class of matrices after those having exactly two entries equal to 1 on each row, already the combinatorial characterization of the minimal conflicting sets is surprisingly complex. Handling the cases of four or five 1s

per row, and so on, might be possible individually, but we expect that there won't be an obvious pattern to generalize, to get the characterization for k 1s per row.

We therefore forego combinatorial characterization in the following, taking an algorithmic approach instead. We will next describe an algorithm that decides if a given row in a general binary matrix belongs to at least one minimal conflicting set. This algorithm has time complexity that is exponential in the maximum number of entries equal to 1 in the matrix. This algorithm is based on Tucker patterns, and, as far as we know, this is the first time these patterns have been used in an effective algorithmic way.

In Chapter 4, we will describe a method to generate all minimal conflicting sets of a general binary matrix. This method is based on the very general framework of monotone Boolean functions, and more precisely on the dualization of monotone Boolean functions. The drawback of this approach is that it requires also the generation of all maximal sets of rows that have the $C1P$, and there can be an exponential number of such sets even if there are few minimal conflicting sets. We will then describe the backtracking approach to generate all cycles of a graph and outline some possible extensions of this method to generate efficiently all minimal conflicting sets. This chapter also contains experimental results on genomic data, both real and simulated.

Chapter 2

Deciding the $C1P$

In this chapter we present methods for deciding whether a given binary matrix M has the $C1P$. We first present the five "forbidden" matrices, introduced by Tucker, which are such that if M contains one or more of them, then M does not have the $C1P$. These five forbidden patterns are important for our work on minimal conflicting sets in Chapter 3. We then introduce an algorithm, called *partition refinement*, which allows us to easily decide whether M has the $C1P$, and, if it does, the algorithm also produces all valid permutations. Based on the idea of partition refinement, we discuss an alternative algorithm, called the *PQR-tree algorithm*, which not only gives us all valid permutations if M has the $C1P$, but also determines all conflicting subsets of the set of columns, if M does not have the $C1P$. Each such conflicting subset is represented by an R-node and is detected using the incompatibility graph, which is introduced in the last section of this chapter.

2.1 Asteroidal Triples and Forbidden Patterns

In this section, we review some important $C1P$ results of Tucker [41]. We first present the five submatrices that prevent a binary matrix M from having the $C1P$. From the previous chapter, we know that for M with two 1s per row, M does not have the $C1P$ if and only if the bipartite graph $B(M)$ contains a cycle or a claw. We discuss the bipartite graphs of the five minimal patterns, which can be viewed as a generalizations of cycles and/or claws. These five patterns are the foundation of our work in Chapter 3 on the combinatorial characterization of minimal conflicting sets. Observing that the consecutive 1s in each row can be considered as an interval in an interval graph, we then discuss the strong relation between $C1P$ matrices

and interval graphs. Finally, we introduce the concept of an *asteroidal triple* and discuss the relation with the $C1P$, based on a result of Lekkerkerker and Boland [2] linking interval graphs and asteroidal triples.

2.1.1 Forbidden Patterns

Definition 2.1.1 For a given binary matrix M, we define the *configuration* of M to be the set of matrices obtained by permuting the rows and/or columns of M.

Example 2.1.2 Let M and M' be the binary matrix given as:

M :

	c_1	c_2	c_3	c_4	c_5	c_6
r_1	**1**	**1**	0	0	0	0
r_2	0	0	**1**	**1**	0	0
r_3	0	0	0	0	**1**	**1**
r_4	0	**1**	0	**1**	0	**1**

M' :

	c_2	c_4	c_6	c_1	c_3	c_5
r_4	**1**	**1**	**1**	0	0	0
r_3	0	0	**1**	0	0	**1**
r_1	**1**	0	0	**1**	0	0
r_2	0	**1**	0	0	**1**	0

We can view M' as a result of permuting some rows and columns of M, hence M' is a member of the configuration of M.

We now define five matrices $M_{I_n}, M_{II_n}, M_{III_n}, M_{IV}$ and M_V, where $1 \leq n < \infty$:

M_{I_n} :

	c_1	c_2	c_3	c_4	...	c_n	c_{n+1}	c_{n+2}
r_1	**1**	**1**	0	0	...	0	0	0
r_2	0	**1**	**1**	0	...	0	0	0
r_3	0	0	**1**	**1**	...	0	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
r_n	0	0	0	0	...	**1**	**1**	0
r_{n+1}	0	0	0	0	...	0	**1**	**1**
r_{n+2}	**1**	0	0	0	...	0	0	**1**

M_{II_n} :

	c_1	c_2	c_3	c_4	...	c_n	c_{n+1}	c_{n+2}	c_{n+3}
r_1	**1**	**1**	0	0	...	0	0	0	0
r_2	0	**1**	**1**	0	...	0	0	0	0
r_3	0	0	**1**	**1**	...	0	0	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
r_n	0	0	0	0	...	**1**	**1**	0	0
r_{n+1}	0	0	0	0	...	0	**1**	**1**	0
r_{n+2}	**1**	**1**	**1**	**1**	...	**1**	**1**	0	**1**
r_{n+3}	0	**1**	**1**	**1**	...	**1**	**1**	**1**	**1**

M_{III_n} :

	c_1	c_2	c_3	c_4	...	c_n	c_{n+1}	c_{n+2}	c_{n+3}
r_1	**1**	**1**	0	0	...	0	0	0	0
r_2	0	**1**	**1**	0	...	0	0	0	0
r_3	0	0	**1**	**1**	...	0	0	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
r_n	0	0	0	0	...	**1**	**1**	0	0
r_{n+1}	0	0	0	0	...	0	**1**	**1**	0
r_{n+2}	0	**1**	**1**	**1**	...	**1**	**1**	0	**1**

M_{IV} :

	c_1	c_2	c_3	c_4	c_5	c_6
r_1	**1**	**1**	0	0	0	0
r_2	0	0	**1**	**1**	0	0
r_3	0	0	0	0	**1**	**1**
r_4	0	**1**	0	**1**	0	**1**

M_V :

	c_1	c_2	c_3	c_4	c_5
r_1	**1**	**1**	0	0	0
r_2	**1**	**1**	**1**	**1**	0
r_3	0	0	**1**	**1**	0
r_4	**1**	0	0	**1**	**1**

Theorem 2.1.3 *[41] A binary matrix M has the C1P if and only if M does not contain a sub-matrix which is a member of the configurations of $M_{I_n}, M_{II_n}, M_{III_n}, M_{IV}$, M_V, where $1 \leq n < \infty$.*

The bipartite graphs for Tucker's five configurations are illustrated below.

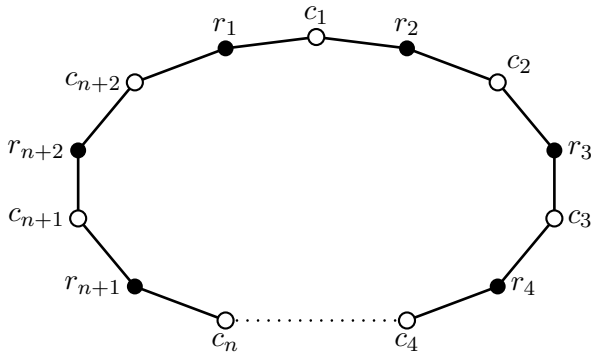


Figure 2.1: $B(M_{I_n})$

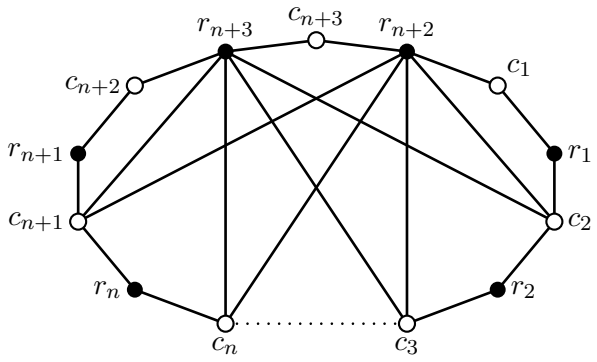


Figure 2.2: $B(M_{II_n})$

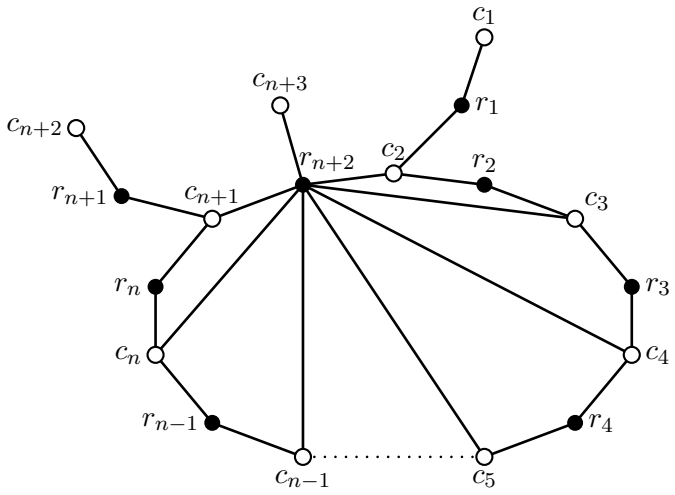


Figure 2.3: $B(M_{III_n})$

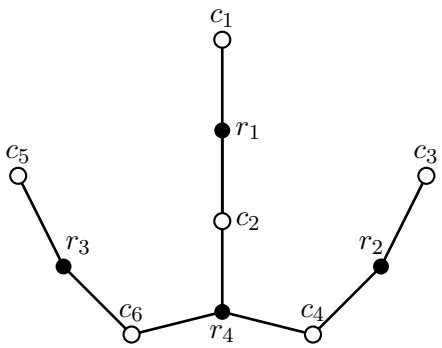


Figure 2.4: $B(M_{IV})$

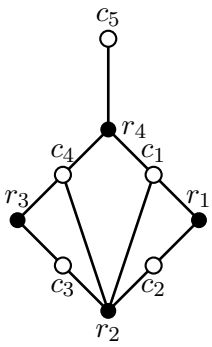


Figure 2.5: $B(M_V)$

The following proposition follows immediately from Theorem 2.1.3:

Proposition 2.1.4 *A binary matrix M has the C1P if and only if $B(M)$ contains none of the graphs $B(M_{I_n}), B(M_{II_n}), B(M_{III_n}), B(M_{IV}), B(M_V)$ as an induced subgraph.*

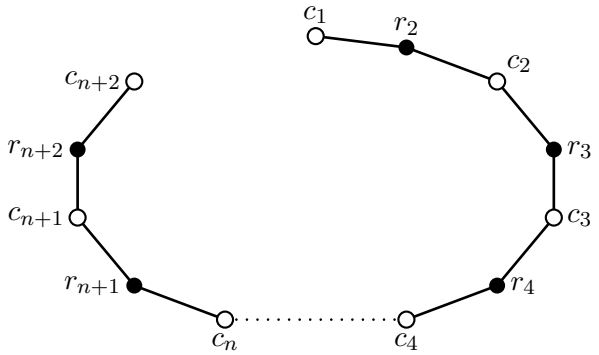
We see that $B(M_{I_n})$ is exactly a cycle, and $B(M_{IV})$ can be viewed as a generalization of a claw, where the root column breaks into three columns c_2, c_4, c_6 which all connect to a newly introduced row r_4. Each of $B(M_{II_n}), B(M_{III_n})$ and $B(M_V)$ can be viewed as a "forced-claw". We discuss this notion by using $B(M_{II_n})$ as an example. If r_1 and r_{n+3} were the only rows in the matrix, the given order of the columns would already be a valid permutation. In particular this would involve the column interval $[c_3, c_4, \ldots, c_{n+1}, c_{n+2}, c_{n+3}]$. Taking row r_{n+2} into account however, we see that columns c_{n+2} and c_{n+3} must be transposed in order to preserve a valid permutation, leaving us with the column interval $[c_3, c_4, \ldots, c_{n+1}, c_{n+3}, c_{n+2}]$. Considering row r_{n+1} then motivates the transposition of c_{n+1} and c_{n+3}, so that we now have $[c_3, c_4, \ldots, c_{n+3}, c_{n+1}, c_{n+2}]$. Continuing this process in order to accommodate rows $r_n, r_{n-1}, \ldots, r_3$, we are eventually forced to have

$[c_{n+3}, c_3, c_4, \ldots, c_{n+1}, c_{n+2}]$. At this point the column neighbors of c_2 are c_1 and c_{n+3}. On the other hand, rows r_1 and r_2 also force c_1 and c_3 to be the two column neighbors of c_2. We thus have a "forced-claw", which arose because of the cyclic structure in the graph whereby every row influenced the two neighboring rows, including r_{n+3} and r_1 having an influence on each other. In this case the root of the claw was c_2, but, again because of the cyclic structure in the graph, any of $c_3, c_4, \ldots, c_{n+1}$ could have been the root instead.

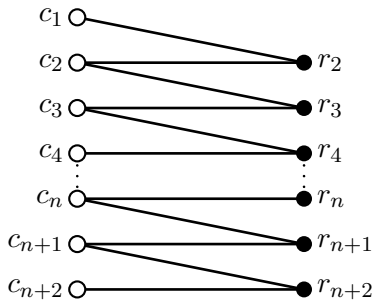
By similar processes, $B(M_{III_n})$ and $B(M_V)$ can also be viewed as "forced-claw" graphs, due to similar cyclic structures. It thus appears that the two kinds of obstructing patterns that prevent M from having the $C1P$ are variants of cycles and claws.

We mentioned earlier that these five configurations are minimal structures that prevent a binary matrix from having the $C1P$. In fact, the set of row vertices in each of the five bipartite graphs forms a minimal conflicting set. For, to begin with, Proposition 2.1.4 shows that these five graphs do not have the $C1P$. Furthermore, as we examine below, removing any row from any of these graphs allows a V_1-consecutive arrangement of the columns, so that the resulting graphs have the $C1P$.

1. Removing any row from $B(M_{I_n})$, say r_1 in the figure, the resulting bipartite graph is:

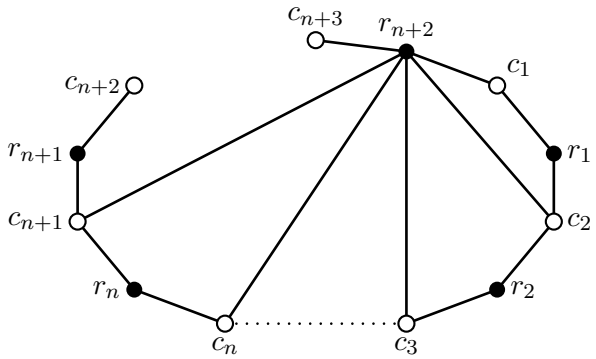


which we can rearrange to get a V_1 consecutive arrangement as shown:

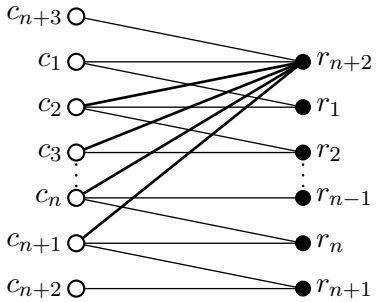


2. There are three types of row vertex we can remove for $B(M_{II_n})$:

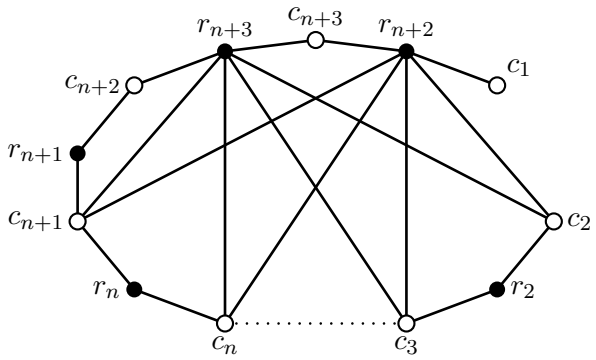
   (a) Removing r_{n+2} or r_{n+3}, say r_{n+3} in the figure, the resulting bipartite graph is:



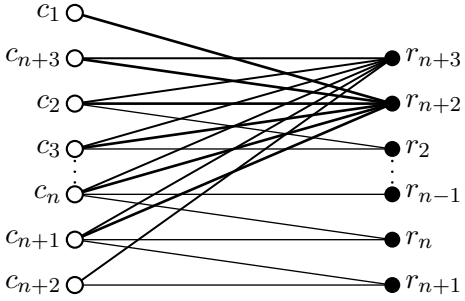
   which can be rearranged to give a V_1 consecutive arrangement as shown:



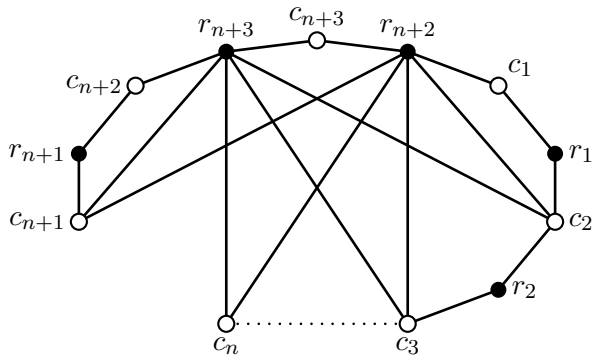
   (b) Removing r_1 or r_{n+1}, say r_1 in the figure, the resulting bipartite graph is:



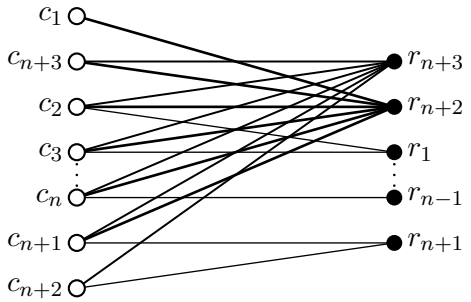
   which can be rearranged to give the following V_1 consecutive arrangement:



(c) Removing one of $r_2, r_3, \ldots, r_n$, say r_n in the figure, the resulting bipartite graph is:

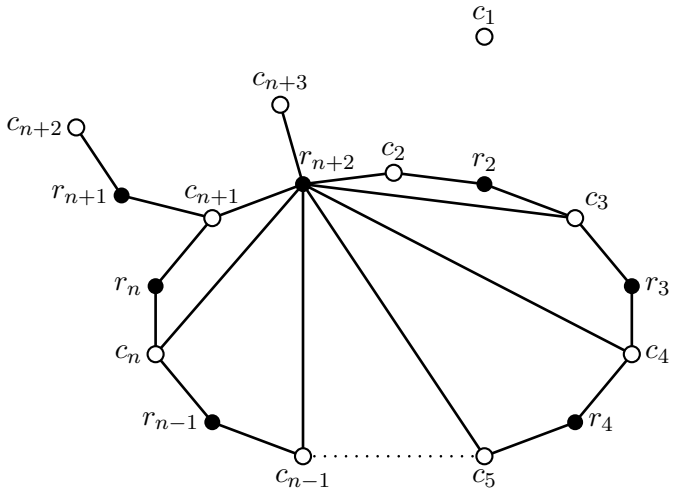


and we can rearrange this to get a V_1-consecutive arrangement as shown:

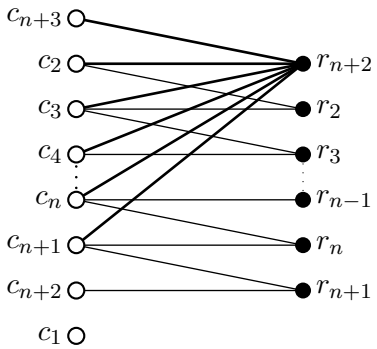


3. For $B(M_{III_n})$, removing r_{n+2} results in a graph similar to that of M_{I_n}, and it obviously has a V_1-consecutive arrangement. There are also three other types of row vertices that we can remove.

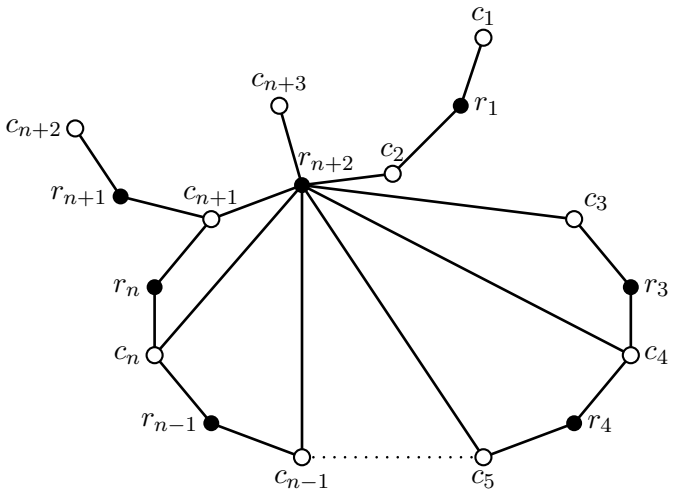
(a) Removing r_1 or r_{n+1}, say r_1 in the figure, the resulting bipartite graph is:



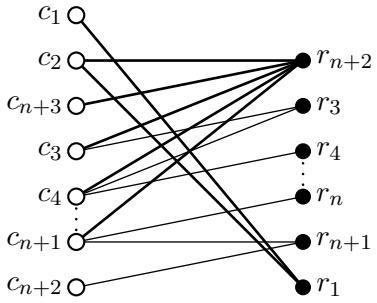
and we can rearrange this graph to get the following V_1-consecutive arrangement:



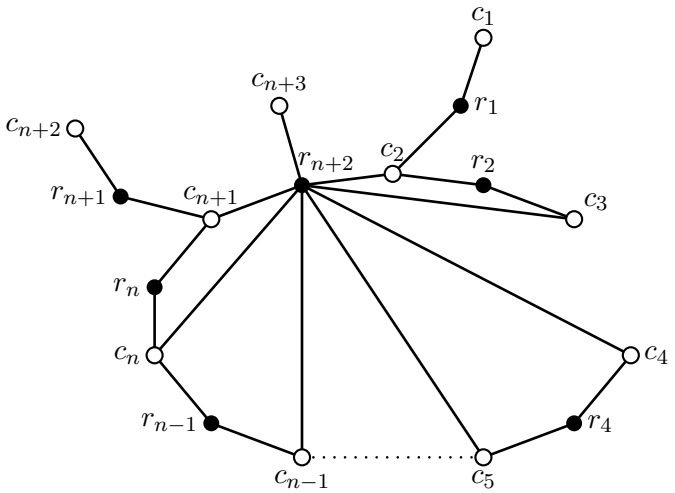
(b) Removing one of r_2 or r_n, say r_2 in the figure, the resulting bipartite graph is:



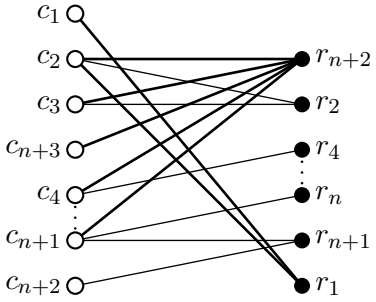
which can be rearranged to give a V_1-consecutive arrangement as shown:



(c) Removing one of $r_3, r_4, \ldots, r_{n-1}$, say r_2 in the figure, the resulting bipartite graph is:



which can be rearranged to give a V_1-consecutive arrangement as shown:



4. Removing any row from $B(M_{IV})$ clearly gives a V_1-consecutive arrangement for the set of columns. For $B(M_V)$, removing r_2 or r_4 gives a V_1 consecutive arrangement. It takes a moment to see that removing either r_1 or r_3 also gives a V_1 consecutive arrangement.

The following proposition follows immediately from the above discussion:

Proposition 2.1.5 *Let M be any of the configurations of $M_{I_n}, M_{II_n}, M_{III_n}, M_{IV}$ or M_V, then $R(M)$ is a minimal conflicting set.*

Therefore the set of rows in each of the five patterns Tucker described is an MCS. However, if M contains more than one of these configurations, the set of rows of M possibly is not an MCS. Such an example is given below:

Example 2.1.6

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
r_1	**1**	0	0	0	**1**	**1**	0	0
r_2	**1**	**1**	**1**	0	**1**	0	**1**	0
r_3	0	**1**	**1**	0	0	0	**1**	**1**
r_4	**1**	0	**1**	**1**	0	**1**	0	**1**

We see that the sub-matrix M_1 consisting of all the rows and columns c_1, c_2, c_3, c_4 and c_5 is a member of configuration of M_V; and the sub-matrix M_2 consisting of all the rows and columns c_5, c_6, c_7 and c_8 is a member of configuration of M_{I_2}. However the set of rows is not an MCS as one can check that removing r_1 does not give a valid permutation. Furthermore the sub-matrix consisting of r_2, r_3, r_4 and c_1, c_7, c_8 is M_{I_1}. We discuss more about this in Chapter 3.

2.1.2 Asteroidal Triples

There is a close relation between $C1P$ matrices and interval graphs: for a given binary matrix M, if M has the $C1P$, then the 1s in each row are consecutive, they form an interval and the $C1P$ for M implies $O(M)$ is an interval graph.

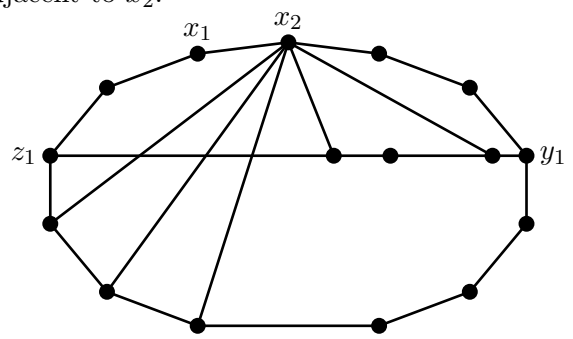
Proposition 2.1.7 *For a binary matrix M with more than three rows, M has the $C1P$ if and only if $O(M)$ is an interval graph.*

Lekkerkerker and Boland [2] give a characterization of interval graphs by using the concept of an asteroidal triple. Since interval graphs and the $C1P$ are strongly related, we take the time to introduce some properties of asteroidal triples, and present some results that lead to a characterization of the $C1P$ in terms of them.

Definition 2.1.8 Given a graph $G(V, E)$ and three distinct vertices $x, y, z \in V(G)$, we say x, y, z form an *asteroidal triple* if there is a path p between any two vertices of $S = \{x, y, z\}$ and no vertex on p is adjacent to the third vertex in S.

Example 2.1.9 Let G be the graph illustrated below. One can check that for the vertices x_1, y_1, z_1 in $V(G)$, we can find three paths p_1, p_2 and p_3 such that x_1, y_1 are on p_1 but no vertex on p_1 is adjacent to z_1; y_1, z_1 are on p_2 but no vertex on p_2 is adjacent to x_1; x_1, z_1 are on P_3 but no vertex on p_3 is adjacent to y_1. Hence x_1, y_1, z_1 form an asteroidal triple.

However, x_2, y_1, z_1 does not form an asteroidal triple, since on any path containing y_1 and z_1, there is some vertex adjacent to x_2.



Theorem 2.1.10 *[2] A graph G is an interval graph if and only if it contains no asteroidal triple and no chordless cycle of length greater than 3.*

Theorem 2.1.11 *[41] For a given binary matrix M, let V_1 be the set of column-vertices in the bipartite graph $B(M)$ of M. Then M has the $C1P$ if and only if V_1 does not contain asteroidal triple.*

In Chapter 3 we will use Tucker's five forbidden configurations in our combinatorial characterization of $C1P$ matrices. To our knowledge, the only algorithm to test for the presence of Tucker's configurations as submatrices of a given matrix is given by Dom**??** in 2008. In the remainder of Chapter 2, we introduce some existing algorithms that can be used to test whether a given matrix has the $C1P$. These are important for us, since some of our own algorithms rely on an oracle that decides the $C1P$.

2.2 Partition Refinement and the $C1P$

In this section, we describe a simple and efficient algorithm which determines whether M has the $C1P$. This algorithm is based on a widely used algorithmic technique called *partition refinement* [22]. Partition refinement deals with a finite set S and a current set partition of S. At each step of the algorithm, it aims to refine the current set partition according to a subset of S. In this section, we use the definition of the $C1P$ in which rows are viewed as sets, so that the partition refinement algorithm will decide whether there is a permutation for the set of columns such that each row in $R(M)$ appears contiguous in this permutation.

2.2.1 General Partition Refinement

For a given set S and a set partition P of S, the general partition refinement algorithm refines P by a given set T and results a new set partition P' of $S \cup T$. In this subsection, we start with some definitions, and then present the procedure of the partition refinement algorithm.

Definition 2.2.1 For a finite set S, we define an *ordered set partition*, P of S to be a linear ordered collection of disjoint subsets of S whose union is S. Each of the disjoint subsets is called a *part* of P.

From now, we use partition or set partition for ordered set partition.

Example 2.2.2 If $S = \{1, 2, 3, 4, 5, 6\}$, then $P_1 = (\{1, 4, 6\}, \{2, 5\}, \{3\})$ is a set partition of S with three parts $\{1, 4, 6\}, \{2, 5\}$ and $\{3\}$. The set partition $(\{4, 6, 1\}, \{2, 5\}, \{3\})$ is considered the same as P_1, but $P_2 = (\{2, 5\}, \{1, 4, 6\}, \{3\})$ is different. Neither $(\{1, 4\}, \{2, 5\}, \{3\})$ nor $(\{1, 4, 6\}, \{2, 5\}, \{3, 6\})$ is a set partition of S.

Definition 2.2.3 Given two set partitions P and P' of S, we write $P' \preccurlyeq P$ if every part of P' is a subset of a single part of P; otherwise we write $P' \not\preccurlyeq P$.

Example 2.2.4 Let S, P_1, P_2 be as given in Example 2.2.2. If $P'_1 = (\{1, 4\}, \{6\}, \{2, 5\}, \{3\})$ and $P'_2 = (\{2, 5\}, \{1, 4, 6, 3\})$, then P'_1 and P'_2 are set partitions of S and $P'_1 \preccurlyeq P_1$, but $P'_2 \not\preccurlyeq P_2$.

Definition 2.2.5 For a set partition P of S and a given $S' \subset S$, we define *the sub-partition $P_{S'}$ of P according to S'* to be the set partition of S' that is induced by P in the following sense: If we write $P = (p_1, p_2, \ldots, p_t)$, then $P_{S'} = (p_1 \cap S', p_2 \cap S', \ldots, p_k \cap S')$.

Example 2.2.6 Let $S = \{1, 2, 3, 4, 5, 6\}$, $P = (\{1, 4, 6\}, \{2, 5\}, \{3\})$ and $S' = \{1, 2, 3, 5\} \subset S$. Then the sub-partition of P according to S' is $P_{S'} = (\{1\}, \{2, 5\}\, \{3\})$.

Given a set partition P of S, and any other set T, we now define what it means to *refine* P by T. Refining P by T consists in finding the set partition P' of $T \cup S$, if it exists, which satisfies:

1. $P'_S \preccurlyeq P$.

2. There does not exist a set partition $P'' \neq P'$ such that $P'_S \preccurlyeq P''_S \preccurlyeq P$.

In fact such a P' always exists and can be found by using the general partition refinement algorithm.

Example 2.2.7 Let $S = \{1,2,3,4,5,6\}$, $T = \{1,2,3,5,7\}$ and $P = (\{1,4,6\},\{2,5\},\{3\})$. Then the refinement of P by T is the set partition $P' = (\{4,6\},\{1\},\{2,5\},\{3\},\{7\})$. The set partition $P'' = (\{4,6\},\{1\},\{2\},\{5\},\{3\},\{7\})$ satisfies condition 1, but it fails condition 2, since $P''_s \preccurlyeq P'_s \preccurlyeq P$.

We now present the general partition refinement procedure[22]:

Let $P = (p_1, p_2, \ldots, p_t)$ be a set partition of S, and let T be a given arbitrary set. We refine P by T and compute a set partition P' of $S \cup T$ according to the following cases:

1. $T \cap S = \emptyset$: In this case $P' = (p_1, p_2, \ldots, p_t, T)$.

2. $T \subseteq S$: The parts in P that are contained in or do not intersect T remain unchanged. We refine the parts that intersect and are not contained in T by splitting each of these parts p_i to two parts: $p_i \cap T$ and $p_i \backslash T$. For each part p_i to be refined, we update P to $P' = (p'_1, p'_2, \ldots, p'_{t+1})$ as follows:

   (a) If $i = 1$, then $P = (p_1 \backslash T, p_1 \cap T, p_2, p_3, \ldots, p_t)$.

   (b) If $i > 1$ and $p_{i-1} \cap T = \emptyset$, then $P = (p_1, p_2, \ldots, p_{i-1}, p_i \backslash T, p_i \cap T, p_{i+1}, p_{i+2}, \ldots, p_t)$.

   (c) If $i > 1$ and $p_{i-1} \cap T \neq \emptyset$, then $P = (p_1, p_2, \ldots, p_{i-1}, p_i \cap T, p_i \backslash T, p_{i+1}, p_{i+2}, \ldots, p_t)$.

3. $T \not\subseteq S$ and $T \cap S \neq \emptyset$:

   (a) If p_1 is the only part of P that intersects T, then $P' = (T \backslash S, T \cap p_1, p_1 \backslash T, p_2, ..., p_t)$.

   (b) Otherwise, we follow the same procedure as in the previous case to refine P by $T \cap S$ and achieve a new set partition $P'' = (p'_1, p'_2, \ldots, p'_v)$ for S. If $p'_1 \cap T \neq \emptyset$, then we take $P' = (T \backslash S, p'_1 \cap T, p'_1 \backslash T, p'_2, \ldots, p'_v)$; otherwise $P' = (p'_1, p'_2, \ldots, p'_v \backslash T, p'_v \cap T, T \backslash S)$.

The general partition refinement procedure updates a given set partition P by a given set T and returns a new set partition P'. This problem can be solved in time $O(|T \cap S|)$ [22].

2.2.2 Using Partition Refinement to Decide the $C1P$

In this subsection, we present a way to use the partition refinement algorithm to decide whether a given binary $m \times n$ matrix M has the $C1P$. We let $\{1, 2, \ldots, n\}$ represent the set of columns of M and $O(M)$ be the overlap graph M.

Definition 2.2.8 Given a set partition $P = (p_1, p_2, \ldots, p_t)$ of S and an arbitrary set T, we say T is consecutive in P if one of the following three cases applies:

1. $T \cap S = \emptyset$: T is consecutive in P automatically.
2. $T \subseteq S$: Let ℓ be the smallest integer such that $p_\ell \cap T \neq \emptyset$ and r be the largest integer such that $p_r \cap T \neq \emptyset$. We require $(p_\ell \cap T) \cup p_{\ell+1} \cup p_{\ell+2} \ldots \cup p_{r-1} \cup (p_r \cap T) = T$.
3. $T \not\subseteq S$ and $T \cap S \neq \emptyset$: Let ℓ be the smallest integer such that $p_\ell \cap T \neq \emptyset$ and r be the largest integer such that $p_r \cap T \neq \emptyset$. We require $(p_\ell \cap T) \cup p_{\ell+1} \cup p_{\ell+2} \ldots \cup p_{r-1} \cup (p_r \cap T) \subset T$ and at least one of the following is true:
   (a) $\ell = 1$ and $p_\ell \subseteq T$.
   (b) $r = t$ and $p_r \subseteq T$.

Example 2.2.9 Let $S = \{1, 2, 3, 4, 5, 6\}, P_1 = (\{1, 4, 6\}, \{2, 5\}, \{3\}), P_2 = (\{2, 5\}, \{1, 4, 6\}, \{3\})$ be as given in Example 2.2.2 and $T = \{1, 2, 3, 5\} \subset S$. Then T is consecutive in P_1 since we can arrange the elements in the parts of P_1 as $(\{4, 6, 1\}, \{2, 5\}, \{3\})$. However, T is not consecutive in P_2 since no matter how we arrange the elements in the parts of P_2, we still have $\{1, 4, 6\}$ as the middle part, and T intersects the first and last part of P_2, but the middle part is not contained in T. Moreover, if $T' = \{1, 2, 7\}$, then T' is not consecutive in P_1 as we are now in case 3(a) and $\{1, 4, 6\} \not\subseteq T'$.

We add at the end one step to the partition refinement procedure, which is to check whether T is consecutive in the updated P'. If this is the case, then we return *success*, otherwise we return *failure*.

Example 2.2.10 For example, suppose we have $S = \{1, 2, 3, 4, 5, 6\}$, $P = (\{1, 3, 4\}, \{6\}, \{2, 5\})$ and we try to refine P by $T = \{1, 2\}$. After the refining process, we have the updated set partition $P' = (\{3, 4\}, \{1\}, \{6\}, \{2\}, \{5\})$. Since $T = \{1, 2\}$ is not consecutive in P', we return *failure*. If instead we try to refine P by $T' = \{1, 2, 6\}$, we will get the same updated P', but now $T' = \{1, 2, 6\}$ is consecutive in P', and we return *success*.

In Algorithm 2.1, we give the procedure for deciding whether a given $R(M)$ has the $C1P$. The algorithm iterates over the components C of the overlap graph $O(M)$, and on each iteration uses a function $SpanTree$, which takes a component C and a vertex r in C as arguments, grows a spanning tree of C rooted at r, using depth-first search, and returns the list $L = (r_{\ell_1}, r_{\ell_2}, \ldots, r_{\ell_t})$ of rows on the tree, in the order in which they were added. We note that the list L returned by $SpanTree$ will always have the property that between any r_{ℓ_i} and r_{ℓ_j} with $i < j$, we can find a path p in C such that for all r_{ℓ_k} on p, we have $k < j$.

Algorithm 2.1 Partition Refinement for the $C1P$

```
1:  for each component C of O(M)
2:      Let t be the number of row vertices in C
3:      if t = 1 then goto next C
4:       else
5:           Pick any vertex r in C
6:           Let (r_{ℓ_1}, r_{ℓ_2}, ..., r_{ℓ_t}) = SpanTree(C, r)
7:           Let P_1 = (r_{ℓ_1})
8:            for i from 2 to t
9:                Let P_i be the set partition resulting from refining P_{i-1} by r_{ℓ_i}.
10:               if r_{ℓ_i} is not consecutive in P_i then return failure
11: return success
```

We note that if the algorithm returns *success*, then the final partition we get is a set partition of the set of columns of M. This partition gives all the valid permutations of M.

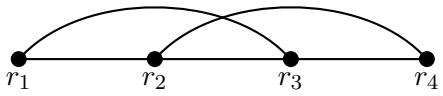
Theorem 2.2.11 *The set of rows of a component in $O(M)$ has the $C1P$ if and only if Algorithm 2.2.2 returns* success.

Proof. From the procedure of Algorithm 2.2.2, we see that we get *success* if and only if we have all rows appearing consecutive in the final set partition. This is exactly what we require for the set of rows of C to have the $C1P$. □

Example 2.2.12 Let M be the binary matrix illustrated below:

	c_1	c_2	c_3	c_4	c_5	c_6	c_7
r_1	**1**	**1**	0	0	**1**	0	**1**
r_2	**1**	**1**	0	0	0	**1**	0
r_3	**1**	0	**1**	0	0	**1**	0
r_4	0	0	**1**	**1**	0	**1**	0

Then $r_1 = \{1,2,5,7\}, r_2 = \{1,2,6\}, r_3 = \{1,3,6\}, r_4 = \{3,4,6\}$ and the corresponding overlap graph is:



As we can see, the overlap graph only has one component. Supposing we pick $r = r_2$ as the first vertex in the spanning tree, we may get two possible lists of vertices. In fact we may pick either of these two lists, and get the same result. Say the list we pick is $L = (r_{\ell_1} = r_2, r_{\ell_2} = r_1, r_{\ell_3} = r_3, r_{\ell_4} = r_4)$.

We first let $P_1 = (r_{\ell_1}) = (\{1,2,6\})$. Then, refining P_1 by r_{ℓ_2} gives $P_2 = (\{6\},\{1,2\},\{5,7\})$. Refining P_2 by r_{ℓ_3} gives $P_3 = (\{3\},\{6\},\{1\},\{2\},\{5,7\})$. Finally, refining P_3 by ℓ_4 gives $P_4 = (\{4\},\{3\},\{6\},\{1\},\{2\},\{5,7\})$. This final set partition gives us all the valid permutations for M, namely: 4361257, 4361275, 7521634 and 5721634. The matrix resulting from permutation 7521634 is pictured below:

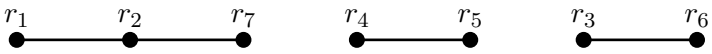
	c_7	c_5	c_2	c_1	c_6	c_3	c_4
r_1	**1**	**1**	**1**	**1**	0	0	0
r_2	0	0	**1**	**1**	**1**	0	0
r_3	0	0	0	**1**	**1**	**1**	0
r_4	0	0	0	0	**1**	**1**	**1**

Recall from Proposition 1.1.13 that a binary matrix M has the $C1P$ if and only if for each component of $O(M)$, its vertex set has the $C1P$. We now give an example of a binary matrix whose overlap graph has more than one components.

Example 2.2.13 Let M be the binary matrix illustrated below:

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}
r_1	**1**	0	0	0	0	0	**1**	0	0	**1**
r_2	0	0	**1**	0	0	**1**	**1**	0	**1**	**1**
r_3	0	0	0	**1**	**1**	0	0	0	0	0
r_4	0	0	**1**	0	0	**1**	0	0	0	0
r_5	0	0	0	0	0	**1**	0	0	**1**	0
r_6	0	**1**	0	**1**	0	0	0	0	0	0
r_7	0	0	**1**	0	0	**1**	0	**1**	**1**	0

Then $r_1 = \{1, 7, 10\}, r_2 = \{3, 6, 7, 9, 10\}, r_3 = \{4, 5\}, r_4 = \{3, 6\}, r_5 = \{6, 9\}, r_6 = \{2, 4\}, r_7 = \{3, 6, 8, 9\}$ and the overlap graph $O(M)$ is:



The overlap graph has three components, and we apply partition refinement on each component separately. For component $C_1 = \{r_1, r_2, r_7\}$ we get the partition $P_1 = (\{1\}, \{7, 10\}, \{3, 6, 9\}, \{8\})$; for component $C_2 = \{r_4, r_5\}$ we get the partition $P_2 = (\{3\}, \{6\}, \{9\})$. Since $r_4 \subset r_2$ and $r_5 \subset r_2$, we can combine P_1 and P_2 to get the partition $P_{12} = (\{1\}, \{7, 10\}, \{3\}, \{6\}, \{9\}, \{8\})$. For component $C_3 = \{r_3, r_6\}$, we get partition $P_3 = (2, 4, 5)$. Since the rows in C_3 are disjoint from the rows in C_1 and C_2, P_{12} and P_3 give us all the valid permutations for columns $1, 3, 6, 7, 8, 9, 10$ and $2, 4, 5$ respectively. Hence we have two sets, $S_1 = \{(1, 7, 10, 3, 6, 9, 8), (1, 10, 7, 3, 6, 9, 8), (8, 9, 6, 3, 10, 7, 1), (8, 9, 6, 3, 7, 10, 1)\}$ and $S_2 = \{(2, 4, 5), (5, 4, 2)\}$, and the set of valid permutations for M arise from the combination of any element of S_1 and any element of S_2 in any order. For example, if we pick $1, 7, 10, 3, 6, 9, 8$ from S_1 and $2, 4, 5$ from S_2, then the valid permutations we get are $1, 7, 10, 3, 6, 9, 8, 2, 4, 5$ and $2, 4, 5, 1, 7, 10, 3, 6, 9, 8$. Therefore we have 16 valid permutations for M in this example. The matrix resulting from the valid permutation $5, 4, 2, 8, 9, 6, 3, 10, 7, 1$ is given below.

	c_5	c_4	c_2	c_8	c_9	c_6	c_3	c_{10}	c_7	c_1
r_1	0	0	0	0	0	0	0	**1**	**1**	**1**
r_2	0	0	0	0	**1**	**1**	**1**	**1**	**1**	0
r_3	**1**	**1**	0	0	0	0	0	0	0	0
r_4	0	0	0	0	0	**1**	**1**	0	0	0
r_5	0	0	0	0	**1**	**1**	0	0	0	0
r_6	0	**1**	**1**	0	0	0	0	0	0	0
r_7	0	0	0	**1**	**1**	**1**	**1**	0	0	0

To recap, the procedure consists in applying the partition refinement algorithm to each component of $O(M)$, the overlap graph of a binary $m \times n$ matrix M. If for any component, we cannot make a row consecutive in the corresponding set partition, then M does not have the $C1P$. Otherwise, M has the $C1P$, and the set of valid permutations for M arises from the combination of the set partitions we get for each component. According to [22], this algorithm has time complexity $O(m + n + c)$, where c is the number of 1s in M.

2.3 PQ-Trees and PQR-Trees

In this section, we present another algorithm for testing the $C1P$ of a given binary matrix M. This algorithm applies the concept of PQ-tree and PQR-tree for $R(M)$. The PQ-tree, introduced by Booth [3] in 1975, is a rooted tree with two kind of of nodes, P-nodes and Q-nodes. The children of a P-node are ordered and can be permuted freely, whereas the children of a Q-node are ordered and can only have their order reversed.

In Booth and Lueker's [4] paper, they used PQ-trees to give the first linear algorithm for testing consecutive ones property with complexity $O(m+n+c)$, where m is the number of rows, n is the number of columns and c is the number of 1s in a binary matrix. For binary matrices that do not have the $C1P$, the algorithm returns no PQ-tree at all; in particular, no information is returned as to where among the columns the conflicts arise.

The PQ-tree concept was extended however in 1998 [32] to PQR-tree, with the introduction of a new type of node, the R-node, which serves to record the conflicts in a non-$C1P$ matrix. Therefore the algorithm always returns a tree, and, for non-$C1P$ matrices, at least one R-node is built.

Definition 2.3.1 A PQR-tree with n leaves is a rooted tree with its n leaves labeled from 1 to n and with three kind of internal nodes, namely P-nodes, Q-nodes and R-nodes. Moreover the children of P-nodes and Q-nodes are linearly ordered and:

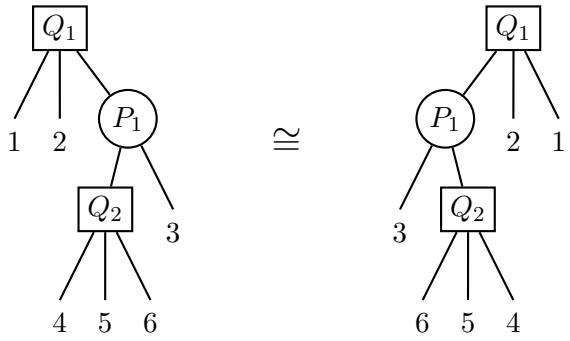
1. P-nodes have at least two children;

2. Q-nodes and R-nodes have at least three children.

A PQ-tree is a PQR-tree without R-nodes.

Definition 2.3.2 Two PQ-trees T, T' are said to be equivalent if one can be obtained from another by permuting the children of P-nodes and/or reversing the order of the children of Q-nodes. We write $T \cong T'$ in this case.

Example 2.3.3 In diagrams, we draw P-nodes as circles, and Q-nodes as rectangles. An example of two equivalent PQ-trees is presented below:



Definition 2.3.4 For a PQ-tree T, we define the equivalence class $E(T)$ of T as $E(T) = \{T' : T \cong T'\}$.

We first describe how to compute a PQR-tree PQR_M from an $m \times n$ binary matrix M, using the overlap graph $O(M)$ as a guide and the partition refinement as the central tool to label internal nodes as P-nodes, Q-nodes or R-nodes. McConnell, in [31] shows that this computation can be done in time $O(n + m + c)$ where c is the number of 1s in M.
The procedure of computing PQR_M from M:

1. Let $C_1, C_2, ..., C_t$ be the components of $O(M)$ and let R_i be the union of rows in C_i, $1 \leq i \leq t$.

2. For i from 1 to t, applying the partition refinement to the rows of C_i, we get a set partition $P_i = (p_{i_1}, \ldots, p_{i_{k_i}})$ of R_i.

3. Let $\mathcal{N} = \{N_1, \ldots, N_q\}$ be the set consists of all the R_is, all the parts p_{i_j} in P_is and $\{1\}, \ldots, \{n\}$.

4. Let PQR_M be the PQR-tree of $\mathcal{N}$: each node is labeled by i $(i = 1 \ldots, q)$ in such a way that i is a child of j if and only if $N_j \subset N_i$ and there does not exists N_k such that $N_j \subset N_k \subset N_i$. Hence, each node N of PQR_M corresponds to a unique subset of $\{1, \ldots, n\}$.

5. For every internal node N of PQR_M, label N as follows:

   (a) R-node if $N = R_i$ for some $i \in \{1, \ldots, t\}$ and C_i does not have the $C1P$,

   (b) Q-node if $N = R_i$ for some i, C_i has the $C1P$ and $k_i > 2$, (i.e. the partition associated to R_i has more than two parts),

   (c) P-node if $N = R_i$ for some i, C_i has the $C1P$, and $k_i \leq 2$,

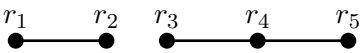
   (d) P-node if $N \neq R_i$ for some i (i.e. N is a part of some P_i).

6. Let PQR_M denote the resulting tree.

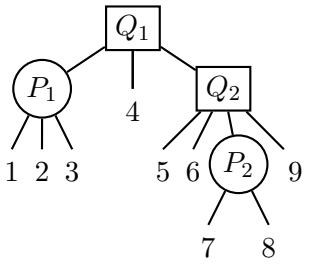
Example 2.3.5 We construct PQR_M of the $C1P$ binary matrix M shown below:

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9
r_1	**1**	**1**	**1**	**1**	0	0	0	0	0
r_2	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**
r_3	0	0	0	0	**1**	**1**	**1**	**1**	0
r_4	0	0	0	0	0	0	**1**	**1**	**1**
r_5	0	0	0	0	0	**1**	**1**	**1**	0

We begin by computing the overlap graph $O(M)$:



which has two components C_1, C_2, where C_1 contains $r_1 = \{1,2,3,4\}, r_2 = \{4,5,6,7,8,9\}$ and C_2 contains $r_3 = \{5,6,7,8\}, r_4 = \{7,8,9\}, r_5 = \{6,7,8\}$. Hence $R_1 = \{1,2,3,4,5,6,7,8,9\}$ and $R_2 = \{5,6,7,8,9\}$. Applying partition refinement to C_1 and C_2, we get $P_1 = (\{1,2,3\}, \{4\}, \{5,6,7,8,9\})$ and $P_2 = (\{5\}, \{6\}, \{7,8\}, \{9\})$. Then $\mathcal{N} = \{\{1\}, \ldots, \{9\}, \{1,2,3\}, \{5,6,7,8,9\}, \{7,8\}\{1,2,3,4,5,6,7,8,9\}\}$. Since $\{1,2,3\}$ and $\{7,8\}$ are the only sets of size greater than 1 that do not equal to R_1 or R_2, they will be labeled as P-nodes. Since P_1 and P_2 have more than two parts, we then label the nodes corresponding to R_1 and R_2 as Q-nodes. This gives us PQR_M as:



Note that, in this example, M has the $C1P$, so PQR_M does not have an R-node and is then a PQ-tree. The leaf orders of the trees in $E(PQR_M)$ represent all valid permutations for M. Indeed, due to the fact that R-nodes are consequences of a component of the overlap graph that does not have the $C1P$, we have the following fundamental result.

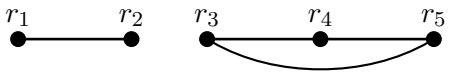
Theorem 2.3.6 *A binary matrix M has the $C1P$ if and only if PQR_M does not have any R-nodes. Moreover, for a $C1P$ matrix M, a permutation is valid for M if and only if it is the leaf order of some member in $E(PQR_M)$.*

We now give an example for a non-$C1P$ matrix.

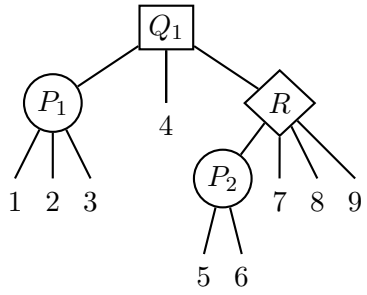
Example 2.3.7 One can check that the following matrix does not have the $C1P$ as we cannot make the $1s$ in r_3,r_4,r_5 consecutive.

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9
r_1	**1**	**1**	**1**	**1**	0	0	0	0	0
r_2	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**
r_3	0	0	0	0	**1**	**1**	**1**	0	0
r_4	0	0	0	0	0	0	**1**	**1**	0
r_5	0	0	0	0	0	0	**1**	0	**1**

this matrix has the overlap graph $O(M)$ as shown:



there are two components, C_1 defined by r_1 and r_2, and C_2 defined by the three other rows in $O(M)$. By applying partition refinement to each of C_1 and C_2, we get the corresponding partitions $P_1 = (\{1,2,3\},\{4\},\{5,6,7,8,9\})$ and $P_2 = (\{5,6\},\{7\},\{8\},\{9\})$. Since r_5 is not consecutive in P_2, in building the nodes at PQR_M, we label the node corresponding to C_2 as an R-node. This gives the following tree PQR_M:



We note that although C_2 does not have the $C1P$, the columns 5 and 6 are grouped under a P-node, as they do not belong to any part that implies a row is not consecutive in P_2. Hence, some partial information can be extracted from a non-$C1P$ component (such parts are called *block* in [31]).

Thus, if the PQR-tree PQR_M for a binary matrix M does not have any R-nodes, then M has the $C1P$, and we get all the valid permutations for M from the valid rearrangements of the leaves of the tree. If instead PQR_M does have one or more R-nodes, then M does not have the $C1P$, and we know that the problematic columns are the children of the R-nodes.

2.4 Incompatibility Graph

The incompatibility graph is introduced in McConnell's [31] paper, where it is shown that for a binary matrix M, $R(M)$ has the $C1P$ if and only if its incompatibility graph is bipartite. Hence if $R(M)$ does not have the $C1P$, its incompatibility graph, which contains $O(n^2)$ vertices, must contain an odd cycle. They then give an algorithm for testing the $C1P$, which returns an odd cycle of length $O(n)$ if $R(M)$ does not have the $C1P$, where n is the number of columns of M. Moreover, if $R(M)$ does not have the $C1P$, each edge of the cycle returned by their algorithm is labeled with a subset of $R(M)$ that documents the incompatibility. Therefore, along with an answer, their algorithm provides a piece of evidence that allows the answer to be easily checked, and is thus a certifying algorithm.

Definition 2.4.1 Let $(x_1, x_2, \ldots, x_n)$ be a valid permutation of a $C1P$ matrix M. We define $Rel = \{(x_i, x_j) : i < j\}$ to be a consecutive ones relation on $(x_1, x_2, \ldots, x_n)$.

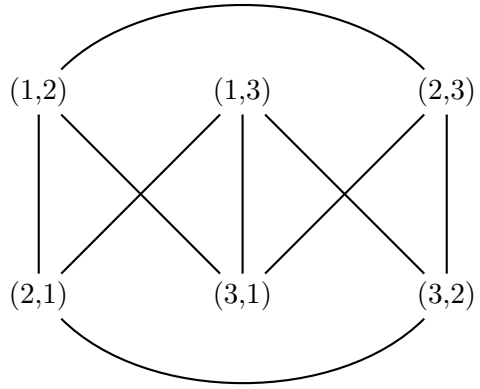
Example 2.4.2 Suppose we have $R(M) = \{\{1, 2, 3\}, \{2, 3\}, \{3, 4\}\}$ for a given M, so that $(1, 2, 3, 4)$ is a valid permutation of M. The consecutive ones relation on this valid permutation is $Rel = \{(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)\}$.

It is easy to see that for an arbitrary pair of columns c_i, c_j, we cannot have (c_i, c_j) and (c_j, c_i) in the same consecutive ones relation. We express this by saying that (c_i, c_j) and (c_j, c_i) are *incompatible*. Similarly, suppose we have $\{c_i, c_j\} \subseteq r_\ell$ and $c_k \notin r_\ell$. Then (c_i, c_k) and (c_k, c_j) cannot appear in the same consecutive ones relation. This is because (c_i, c_k) and (c_k, c_j) implies c_k goes between c_i and c_j in a valid permutation, whereas r_ℓ is not consecutive in this permutation, so we have a contradiction. In this case, we also say that (c_i, c_k) and (c_k, c_j) are incompatible. From these two types of incompatible pairs, we can construct an *incompatibility graph*, where the vertices are all the pairs of columns, and where there is an edge between two pairs if and only if they are incompatible.

Definition 2.4.3 For a given binary matrix M, we define the *simple undirected incompatibility graph* $I(M)$ corresponding to $R(M)$ to be the graph whose vertex set is the set of ordered pairs of columns $\{(c_i, c_j) : 1 \leq i, j \leq n\}$, and in which there is an edge between any two vertices if and only if one of the following holds:

1. The two vertices are of the form $(c_i, c_j), (c_j, c_i)$.
2. The two vertices are of the form $(c_i, c_k), (c_k, c_j)$ provided $\{c_i, c_j\} \subseteq r_\ell$ and $c_k \notin r_\ell$ for some $r_\ell \in R(M)$.

Example 2.4.4 If we have $R(M) = \{\{1, 2\}, \{2, 3\}, \{1, 3\}\}$, then the corresponding incompatibility graph is:



By the definition of the incompatibility graph, a consecutive ones relation must have no incompatible pairs. Since all pairs of the type $(c_i, c_k), (c_k, c_j)$ are symmetric, i.e. (c_i, c_k) and (c_k, c_j) are incompatible if and only if (c_j, c_k) and (c_k, c_i) are incompatible, a consecutive ones relation must consist of half of the vertices of its incompatibility graph and these vertices are independent in this graph. Moreover, the reverse of a valid permutation is also valid, so that the remaining half of the vertices of the incompatibility graph must be independent as well. Now it is clear that if $R(M)$ has the $C1P$, then $I(M)$ must be bipartite. In McConnell's paper [31] it is shown that if $R(M)$ does not have the $C1P$, then $I(M)$ is not bipartite and $I(M)$ contains an odd cycle of length at most $n + 2$.

Theorem 2.4.5 *[31] Let M be a binary matrix with n columns. Then $R(M)$ has the $C1P$ if and only if $I(M)$ is bipartite. If $R(M)$ does not have the $C1P$, then $I(M)$ has an odd cycle of length at most $n + 2$.*

Part II

New Results

Chapter 3

Minimum Conflicting Sets and Tucker Patterns

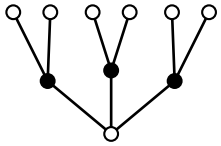
In this chapter, we make use of Tucker patterns to attack two problems. The first one is to define a combinatorial characterization of minimal conflicting sets for matrices with exactly three 1s per row. Next, we describe an algorithm to decide if a given row of a binary matrix belongs to at least one minimal conflicting set.

3.1 Minimal Conflicting Sets in Matrices with Three 1s per Row

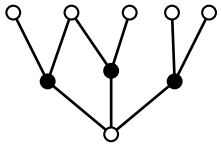
In this chapter, we give a combinatorial characterization of minimal conflicting sets for binary matrices with exactly three 1s per row, based on Tucker's five forbidden patterns. We give all the possible patterns of minimal conflicting sets for matrices with three 1s per row. The patterns are discovered in a naive way in which we try all the possible ways of adding 1s to Tucker's five forbidden patterns. From this characterization, we foresee the difficulty of extending the characterization for matrices with more than three 1s per row.

Definition 3.1.1 We define a bipartite graph to be *claw-like if it is one of* $Claw_1$, $Claw_2$, $Claw_3$, $Claw_4$.

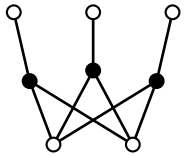
$Claw_1$



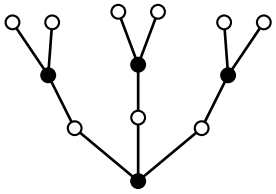
$Claw_2$



$Claw_3$



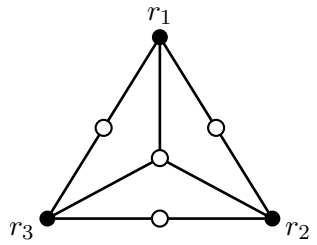
$Claw_4$



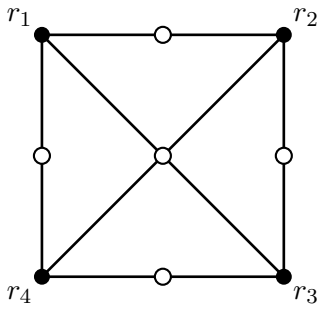
Proposition 3.1.2 *Let M be a binary matrix with three 1s per row.*
If M has exactly three rows, say $R(M) = \{r_1, r_2, r_3\}$, which satisfy $r_1 \cap r_2 \cap r_3 \neq \emptyset$ and $r_i \backslash (r_{i+1 \pmod 3} \cup r_{i+2 \pmod 3}) \neq \emptyset$ for all i, then $B(M)$ must be one of $Claw_1$, $Claw_2$, or $Claw_3$ illustrated in Definition 3.1.1.
If M has exactly four rows, say $R(M) = \{r_1, r_2, r_3, r_4\}$, which satisfy $r_i \cap r_4 \neq \emptyset, r_i \cap r_{i+1 \pmod 3} = \emptyset$ and $r_i \backslash r_4 \neq \emptyset$ for all i, then $B(M)$ must be $Claw_4$ illustrated in Definition 3.1.1.

Definition 3.1.3 We say $B(M)$ is *cycle-like* if it is of the form of one of the graphs $Cycle_1$, $Cycle_2$, $Cycle_3$.

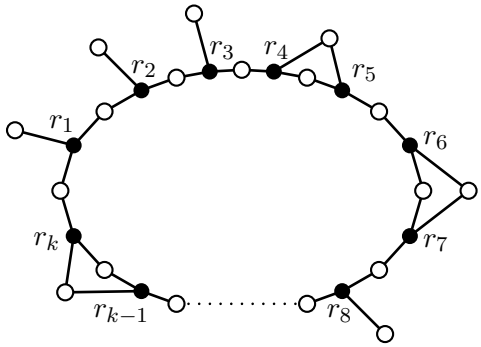
$Cycle_1$:



$Cycle_2$:



$Cycle_3$:



Proposition 3.1.4 *Let M be a binary matrix with three 1s per row.*
If M has exactly three rows, say $R(M) = \{r_1, r_2, r_3\}$, such that $|\cap_{i=1}^{3} r_i| = 1$, and for each

i there is some x_i in $r_i \cap r_{i=1 \pmod 3}$, and none of these x_i's belong to more than two rows, then $B(M)$ is $Cycle_1$ illustrated in Definition 3.1.3.
If M has exactly four rows, say $R(M) = \{r_1, r_2, r_3, r_4\}$, such that $|\cap_{i=1}^{4} r_i| = 1$, and for each i there is some x_i in $r_i \cap r_{i+1 \pmod 4}$, and none of these x_i's belong to more than two rows, then $B(M)$ is $Cycle_2$ illustrated in Definition 3.1.3.
If M has exactly k rows, where $k \geq 3$, say $R(M) = \{r_1, r_2, ...r_k\}$, and the k rows satisfy the condition that whenever $|(i-j) \pmod k| \neq 1$, we have $r_i \cap r_j = \emptyset$, then $B(M)$ is of the form of $Cycle_3$, where $Cycle_3$ stands for the infinite family of bipartite graphs exemplified in Definition 3.1.3.

Lemma 3.1.5 *Let M be one of Tucker's configurations and M^* be a binary matrix that contains M. If $R(M^*)$ is a minimal conflicting set, then M and M^* have the same number of rows.*

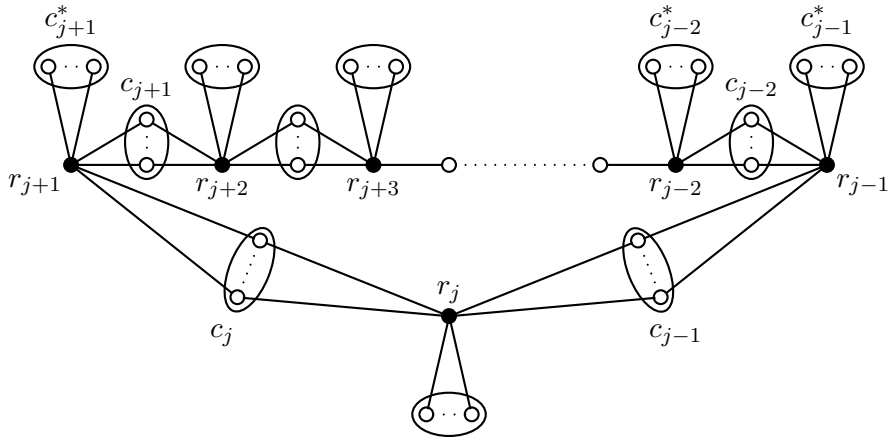
Proof. Since M is a sub-matrix of M^*, M^* has at least as many rows as M. Suppose M has k rows and M^* has more than k rows. Since M is exactly one of the Tucker patterns, $R(M)$ is a minimal conflicting set. Since $R(M) \subsetneq R(M^*)$, there exists a row $r \in R(M^*)$ such that $R(M^*) \backslash r$ does not have the $C1P$, which contradicts that $R(M^*)$ is a minimal conflicting set. □

Proposition 3.1.6 *Let M^* be a binary matrix resulting by appending columns $c_{d_1}, c_{d_2}, \ldots, c_{d_\ell}$ to M_{I_n}, where $n \geq 3$ and $\ell \geq 1$. Then $R(M^*)$ is a minimal conflicting set if and only if each of the appended columns satisfies exactly one of the following conditions:*

1. *the appended column has exactly two 1s and these two 1s belong to two consecutive rows in M^* (the first and the last row are considered to be consecutive as well);*

2. *the appended column has exactly one entry equal to 1.*

Proof. Without loss of generality, for the row indices, we work modulo $n+2$ with $1, 2, \ldots, n+2$ as our system of residues.
($\Leftarrow$) Let $r_1, r_2, \ldots, r_{n+2}$ be the ordered row vertices in $B(M_{I_n})$ such that r_i is only adjacent to r_{i-1} and r_{i+1}. Suppose c_{d_i} contains either two consecutive 1s or a single 1 for all $i \in \{1, 2, \ldots, \ell\}$. Then r_i is only adjacent to r_{i-1} and r_{i+1} in $B(M^*)$. Let $c_i = r_i \cap r_{i+1}$ and $c_i^* = r_i \backslash r_{i-1} \cup r_{i+1}$, $1 \leq i \leq n+2$. We illustrate $B(M^*)$ as:



Now deleting any row r_j from $B(M^*)$, we "break" the cycle, and we can order the columns as $c_j,\ c^*_{j+1},\ c_{j+1} \dots c^*_{j-1},\ c_{j-1}$ to get a valid permutation for $R(M^*) - r_j$. Therefore $R(M^*)$ is a minimal conflicting set.

($\Rightarrow$) Suppose to the contrary that there exists some c_{d_v} containing two non-consecutive 1s. That is, $M^*_{id_v} = M^*_{jd_v} = 1$ for some $i \neq j, j \pm 1$. Then we will see a chord $r_i r_j$ in $B(M^*)$. There are two cases to consider:

1. If there exists some k such that $M^*_{kd_v} = 0$, then $C_1 = (r_i, \dots, r_k, \dots, r_j, d_v)$ contains a cycle in $B(M^*)$. This implies that $R(M^*)\backslash r$, for some r not on C_1, does not have the $C1P$, which contradicts that $R(M^*)$ is a minimal conflicting set.

2. If $M^*_{kd_v} = 1$ for all k, then c_{d_v} is adjacent to all row vertices in $B(M^*)$. Since $n \geq 3$, we have at least five rows and it is possible to pick three non-consecutive row vertices according to $B(M_{I_n})$. Let r_ℓ, r_s, r_t be three non-consecutive row vertices on $B(M_{I_n})$, then we can find column vertices c_ℓ, c_s, c_t such that $c_\ell \in r_\ell \backslash (r_s \cup r_t)$, $c_s \in r_s \backslash (r_\ell \cup r_t)$ and $c_t \in r_t \backslash (r_\ell \cup r_s)$. Hence the subgraph induced on $B(M^*)$ by the vertex subset $\{c_{dv}, r_\ell, r_s, r_t, c_\ell, c_s, c_t\}$ is a claw, which contradicts that $R(M^*)$ is a minimal conflicting set.

□

Example 3.1.7 The following matrices M^{*1} and M^{*2} result from M_{I_3} by appending one column.

$M^{*1} =$

	c_1	c_2	c_3	c_4	c_5	c_6
r_1	**1**	**1**	0	0	0	0
r_2	0	**1**	**1**	0	0	**1**
r_3	0	0	**1**	**1**	0	**1**
r_4	0	0	0	**1**	**1**	0
r_5	**1**	0	0	0	**1**	**1**

We see that c_6 in M^{*1} contains two non-consecutive 1s. For example, (M^{*1}_{26}, M^{*1}_{56}), (M^{*1}_{36}, M^{*1}_{56}) and (M^{*1}_{56}, M^{*1}_{36}) are pairs of non-consecutive 1s. It is easy to see that $R(M^{*1})$ is not a minimal conflicting set as the sub-matrix containing rows r_3, r_4, r_5 and columns c_4, c_5, c_6 is M_{I_1}.

$M^{*2} =$

	c_1	c_2	c_3	c_4	c_5	c_6
r_1	**1**	**1**	0	0	0	**1**
r_2	0	**1**	**1**	0	0	**1**
r_3	0	0	**1**	**1**	0	**1**
r_4	0	0	0	**1**	**1**	**1**
r_5	**1**	0	0	0	**1**	**1**

We see that c_6 in M^{*2} contains all 1s. It is easy to check that a set of any three non-consecutive rows does not have the $C1P$. For example, if we pick $S = \{r_1 = \{1, 2, 6\}, r_3 = \{3, 4, 6\}, r_4 = \{4, 5, 6\}\}$, after applying partition refinement to S, we obtain a set partition $P = (\{1, 2\}, \{6\}, \{4\}, \{3\}, \{5\})$. Since r_3 is not consecutive in P, S does not have the $C1P$. Moreover, a set of any three non-consecutive rows in M^{*2} is a minimal conflicting set and its corresponding bipartite graph is $Claw_2$.

Theorem 3.1.8 *Given a binary matrix M with three 1s per row, $R(M)$ is a minimal conflicting set if and only if $B(M)$ is claw-like or cycle-like.*

Proof.
($\Leftarrow$) Suppose $B(M)$ is claw-like or cycle-like. We check in each of the possible cases that $R(M)$ is a minimal conflicting set. Without loss of generality, we label the columns by positive integers.

1. Claw-like:

   (a) $Claw_1$, $Claw_2$, $Claw_3$: Without loss of generality, we let $r_1 = \{1, 2, x\}, r_2 = \{1, 3, y\}, r_3 = \{1, 4, z\}$, with possibly $x = y, y = z, x = z$ or $x = y = z$. We apply partition refinement to $R(M)$, resulting in a sub-partition $P = (\{2\}, \{1\}, \{3\}, \{4\})$. Since r_3 is not consecutive in P, $R(M)$ does not have the $C1P$.

   It is easy to see that deleting any row, we only have two rows left in $R(M)$ and there is a valid permutation for $R(M)$. Hence $R(M)$ is a minimal conflicting set.

   (b) $Claw_4$: Without loss of generality, we let $r_1 = \{1, 4, x\}, r_2 = \{2, 5, y\}, r_3 = \{3, 6, z\}, r_4 = \{1, 2, 3\}$, where $x \neq y \neq z \neq x$. We apply partition refinement to $R(M)$, resulting in a sub-partition $P = (\{4\}, \{1\}, \{3\}, \{2\}, \{5\}, \{6\})$. Since r_3 is not consecutive in P, $R(M)$ has no valid permutation.

   If we delete r_4 in $R(M)$, we will have three non-overlapping rows, which clearly have the $C1P$. If r_4 remains in $R(M)$, it is not hard to see that if we delete any of the other rows then the remaining rows will have the $C1P$. Therefore $R(M)$ is a minimal conflicting set.

2. Cycle-like:

   (a) $Cycle_1$: Without loss of generality, we let $r_1 = \{1, 2, 4\}, r_2 = \{2, 3, 4\}, r_3 = \{1, 3, 4\}$. We apply partition refinement to $R(M)$, resulting in the set partition $P = (\{1\}, \{4\}, \{2\}, \{3\})$. Since r_3 is not consecutive in P, $R(M)$ does not have the $C1P$.

   Deleting any row from $R(M)$, we have only two rows left, which clearly has the $C1P$. Hence $R(M)$ is a minimal conflicting set.

   (b) $Cycle_2$: Without loss of generality, we let $r_1 = \{1, 2, 5\}, r_2 = \{2, 3, 5\}, r_3 = \{3, 4, 5\}, r_4 = \{1, 4, 5\}$. We apply partition refinement to $R(M)$, resulting in the set partition $P = (\{1\}, \{2\}, \{5\}, \{3\}, \{4\})$. Since r_4 is not consecutive in P, $R(M)$ does not have the $C1P$.

   One can easily check that $R(M) \backslash R_i$ has the $C1P$ for any $i \in \{1, 2, 3, 4\}$. Hence $R(M)$ is a minimal conflicting set.

   (c) $Cycle_3$: This case was dealt with in the first part of the proof of Proposition 3.1.6.

($\Rightarrow$) Suppose $R(M)$ is a minimal conflicting set. We need to show that $B(M)$ is either claw-like or cycle-like. If $R(M)$ is a conflicting set then $R(M)$ does not have the $C1P$, implying that M contains one of Tucker's configuration of matrices, say M'.

By Lemma 3.1.5, all possible minimal conflicting sets must arise from Tucker's configurations by adding columns. We also have the property that each row of M contains exactly three 1s, and therefore we have the strategy of adding some columns to each of Tucker's configurations, to obtain new matrices, and then, for each of these, checking whether the set of rows is a minimal conflicting set. If so, then we need to show that its bipartite graph is either claw-like or cycle-like. If not, then it must contain some minimal conflicting sets, and we then need to show that the bipartite graphs of these minimal conflicting sets are either claw-like or cycle-like.

Since M has exactly three 1s per row, the possible Tucker's configurations we need to consider are: M_{II_1}, M_{III_1}, M_{III_2}, M_{IV} and M_{I_k} for any $k \geq 1$.

1. M_{II_1}:

   (a) Appending one column to M_{II_1}, a resulting matrix M^* is:

	c_1	c_2	c_3	c_4	c_5
r_1	**1**	**1**	0	0	**1**
r_2	0	**1**	**1**	0	**1**
r_3	**1**	**1**	0	**1**	0
r_4	0	**1**	**1**	**1**	0

   We have $r_1 = \{1, 2, 5\}, r_2 = \{2, 3, 5\}, r_3 = \{1, 2, 4\}, r_4 = \{2, 3, 4\}$. Applying partition refinement to $R(M^*)$, we get the set partition $P = (\{4\}, \{1\}, \{2\}, \{5\}, \{3\})$. Since r_4 is not consecutive in P, $R(M^*)$ does not have the $C1P$. It is easy to check that deleting any row from $R(M^*)$, the resulting set has the $C1P$, and therefore $R(M^*)$ is a minimal conflicting set. The corresponding bipartite graph $B(M^*)$ is $Cycle_2$.

   (b) Appending two columns to M_{II_1}, a resulting matrix M^* is:

	c_1	c_2	c_3	c_4	c_5	c_6
r_1	**1**	**1**	0	0	**1**	0
r_2	0	**1**	**1**	0	0	**1**
r_3	**1**	**1**	0	**1**	0	0
r_4	0	**1**	**1**	**1**	0	0

We have $r_1 = \{1, 2, 5\}, r_2 = \{2, 3, 6\}, r_3 = \{1, 2, 4\}, r_4 = \{2, 3, 4\}$, and $R(M^*)$ is not a minimal conflicting set since $R(M^*)\backslash r_4$ does not have the $C1P$. One can check that $R(M^*)\backslash r_3$ and $R(M^*)\backslash r_4$ are both minimal conflicting sets and the corresponding bipartite graph is $Claw_2$.

2. M_{III_1}: Let M^* be a matrix obtained by appending columns to M_{III_1}. Since M^* contains M, it does not have the $C1P$. Deleting any row from M^*, we have two rows left in $R(M^*)$, and this clearly has the $C1P$. Hence regardless of the type of columns we append to M_{III_1}, the set of rows of the resulting matrix M^* is a minimal conflicting set. One can check that, depending on the different types of columns we append, the corresponding bipartite graph is one of $Claw_1$, $Claw_2$, $Claw_3$, and $Cycle_1$.

3. M_{III_2}:

   (a) Appending one column to M_{III_2}, a resulting matrix M^* is:

	c_1	c_2	c_3	c_4	c_5	c_6
r_1	**1**	**1**	0	0	0	**1**
r_2	0	**1**	**1**	0	0	**1**
r_3	0	0	**1**	**1**	0	**1**
r_4	0	**1**	**1**	0	**1**	0

   We have $r_1 = \{1, 2, 6\}, r_2 = \{2, 3, 6\}, r_3 = \{3, 4, 6\}, r_4 = \{2, 3, 5\}$. One can check that $R(M^*)$ is not a minimal conflicting set as $R(M^*)\backslash r_2$ does not have the $C1P$. The set $R(M^*)\backslash r_2$ is a minimal conflicting set and the corresponding bipartite graph is in the $Cycle_3$ family.

   (b) Appending two columns to M_{III_2}, we have three cases:

      i. The resulting matrix M^* is:

	c_1	c_2	c_3	c_4	c_5	c_6	c_7
r_1	**1**	**1**	0	0	0	**1**	0
r_2	0	**1**	**1**	0	0	0	**1**
r_3	0	0	**1**	**1**	0	0	**1**
r_4	0	**1**	**1**	0	**1**	0	0

We have $r_1 = \{1, 2, 6\}, r_2 = \{2, 3, 7\}, r_3 = \{3, 4, 7\}, r_4 = \{2, 3, 5\}$ and $R(M^*)$ is not a minimal conflicting set as $R(M^*)\backslash r_3$ does not have the $C1P$. The set $R(M^*)\backslash r_3$ is a minimal conflicting set and the corresponding bipartite graph is $Claw_2$.

ii. The resulting matrix M^* is:

	c_1	c_2	c_3	c_4	c_5	c_6	c_7
r_1	**1**	**1**	0	0	0	**1**	0
r_2	0	**1**	**1**	0	0	**1**	0
r_3	0	0	**1**	**1**	0	0	**1**
r_4	0	**1**	**1**	0	**1**	0	0

One can check that $R(M^*)$ is not a minimal conflicting set as $R(M^*)\backslash r_1$ does not have the $C1P$. The set $R(M^*)\backslash r_1$ is the only minimal conflicting set and the corresponding bipartite graph is $Claw_2$.

iii. The resulting matrix M^* is:

	c_1	c_2	c_3	c_4	c_5	c_6	c_7
r_1	**1**	**1**	0	0	0	**1**	0
r_2	0	**1**	**1**	0	0	0	**1**
r_3	0	0	**1**	**1**	0	**1**	0
r_4	0	**1**	**1**	0	**1**	0	0

We have $r_1 = \{1, 2, 6\}, r_2 = \{2, 3, 7\}, r_3 = \{3, 4, 6\}, r_4 = \{2, 3, 5\}$ and $R(M^*)$ is not a minimal conflicting set. One can check that $R\backslash R_1$ and $R\backslash R_3$ are both minimal conflicting sets, and their corresponding bipartite graphs are both $Claw_2$. The sets $R\backslash R_2$ and $R\backslash R_4$ are also minimal conflicting sets, and their corresponding bipartite graphs are both in the $Cycle_3$ family.

(c) Appending three columns to M_{III_2}, a resulting matrix M^* is:

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
r_1	**1**	**1**	0	0	0	**1**	0	0
r_2	0	**1**	**1**	0	0	0	**1**	0
r_3	0	0	**1**	**1**	0	0	0	**1**
r_4	0	**1**	**1**	0	**1**	0	0	0

We have $r_1 = \{1, 2, 6\}, r_2 = \{2, 3, 7\}, r_3 = \{3, 4, 8\}, r_4 = \{2, 3, 5\}$ and $R(M^*)$ is not a minimal conflicting set. One can check that $R(M^*)\backslash r_1$ and $R(M^*)\backslash r_3$ are minimal conflicting sets, and the corresponding bipartite graph is $Claw_2$.

4. M_{IV}:

(a) Appending one column to M_{IV}, a resulting matrix M^* is:

	c_1	c_2	c_3	c_4	c_5	c_6	c_7
r_1	**1**	**1**	0	0	0	0	**1**
r_2	0	0	**1**	**1**	0	0	**1**
r_3	0	0	0	0	**1**	**1**	**1**
r_4	0	**1**	0	**1**	0	**1**	0

We have $r_1 = \{1, 2, 7\}, r_2 = \{3, 4, 7\}, r_3 = \{5, 6, 7\}, r_4 = \{2, 4, 6\}$ and $R(M^*)$ is not a minimal conflicting set. One can check that $R(M^*)\backslash r_1$, $R(M^*)\backslash r_2$, $R(M^*)\backslash r_4$ are minimal conflicting sets. The corresponding bipartite graph for $R(M^*)\backslash r_1$, and for $R(M^*)\backslash r_2$ is in the $Cycle_3$ family, and for $R(M^*)\backslash r_4$ is $Claw_1$.

(b) Appending two columns to M_{IV}, a resulting matrix M^* is:

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
r_1	**1**	**1**	0	0	0	0	**1**	0
r_2	0	0	**1**	**1**	0	0	**1**	0
r_3	0	0	0	0	**1**	**1**	0	**1**
r_4	0	**1**	0	**1**	0	**1**	0	0

We have $r_1 = \{1, 2, 7\}, r_2 = \{3, 4, 7\}, r_3 = \{5, 6, 8\}, r_4 = \{2, 4, 6\}$ and $R(M^*)$ is not a minimal conflicting set. One can check that $R(M^*)\backslash r_3$ is a minimal

conflicting set, and the corresponding bipartite graph is in the $Cycle_3$ family. All other cases of appending two columns to make three 1s on each row are similar to this case: The set of four rows is not a minimal conflicting set since the two rows with the 1s on the same column along with the last row always form a minimal conflicting set, and the corresponding bipartite graph is in the $Cycle_3$ family.

(c) Appending three columns to M_{IV}, a resulting matrix M^* is:

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9
r_1	**1**	**1**	0	0	0	0	**1**	0	0
r_2	0	0	**1**	**1**	0	0	0	**1**	0
r_3	0	0	0	0	**1**	**1**	0	0	**1**
r_4	0	**1**	0	**1**	0	**1**	0	0	0

We have $r_1 = \{1, 2, 7\}, r_2 = \{3, 4, 8\}, r_3 = \{5, 6, 9\}, r_4 = \{2, 4, 6\}$. It is easy to check that $R(M^*)$ is a minimal conflicting set and $B(M^*)$ is $Claw_4$.

5. M_{I_n}:

(a) $n = 1$: By the same reason for case M_{III_1}, regardless of the type of columns we append to M_{I_1}, the set of rows of the resulting matrix is a minimal conflicting set.

(b) $n = 2$: If each of the columns appended contains either a single 1 or two consecutive 1s, then the corresponding bipartite graph is in the $Cycle_3$ family. As we have shown in the first part, the set of rows is a minimal conflicting set. We consider now the cases in which at least one of the columns appended either contains exactly two 1s, which are non-consecutive, or contains three or more 1s.

i. Appending one column to M_{I_2}, a resulting matrix is:

	c_1	c_2	c_3	c_4	c_5
r_1	**1**	**1**	0	0	**1**
r_2	0	**1**	**1**	0	**1**
r_3	0	0	**1**	**1**	**1**
r_4	**1**	0	0	**1**	**1**

Notice that if we permute the columns in the order c_1, c_5, c_3, c_4, c_2 and rows in the order r_1, r_2, r_4, r_3, we get the same matrix as the matrix for the first case of configuration M_{II_1}, so this case is the same as $M_{II_1}(i)$.

ii. Appending two columns to M_{I_2}, there are two cases:

A. a resulting matrix M^* is:

	c_1	c_2	c_3	c_4	c_5	c_6
r_1	**1**	**1**	0	0	**1**	0
r_2	0	**1**	**1**	0	0	**1**
r_3	0	0	**1**	**1**	**1**	0
r_4	**1**	0	0	**1**	0	**1**

One can check that any set of three rows of M^* is a minimal conflicting set and its bipartite graph is in the $Cycle_3$ family.

B. a resulting matrix M^* is:

	c_1	c_2	c_3	c_4	c_5	c_6
r_1	**1**	**1**	0	0	**1**	0
r_2	0	**1**	**1**	0	**1**	0
r_3	0	0	**1**	**1**	**1**	0
r_4	**1**	0	0	**1**	0	**1**

One can check that $R(M^*)\backslash r_2$ does not have the $C1P$. The only minimal conflicting set we can get is $S = \{r_1, r_3, r_4\}$ and its bipartite graph is in the $Cycle_3$ family.

iii. Appending three columns to M_{I_2}, a resulting matrix M^* is:

	c_1	c_2	c_3	c_4	c_5	c_6	c_7
r_1	**1**	**1**	0	0	**1**	0	0
r_2	0	**1**	**1**	0	0	**1**	0
r_3	0	0	**1**	**1**	**1**	0	0
r_4	**1**	0	0	**1**	0	0	**1**

One can check that $R(M^*)\backslash r_2$ does not have the $C1P$. The minimal conflicting sets we can get are $S_1 = \{r_3, r_4, r_1\}$ and $S_2 = \{r_1, r_2, r_3\}$ and their bipartite graphs are in the $Cycle_3$ family.

(c) $n \geq 3$: Let M^* be a matrix obtained from M_{I_n} by appending columns. By Proposition 3.1.6, $R(M^*)$ is a minimal conflicting set if and only if each column appended contains either a single 1 or two consecutive 1s. If this is the case, $B(M^*)$ is in the $Cycle_3$ family. If $R(M^*)$ is not a minimal conflicting set, then some of its subsets are minimal conflicting sets. For any of the subsets that is a minimal conflicting set, its corresponding matrix must contain one of Tucker's configurations. Therefore it must be one of the cases we discussed above, and hence its bipartite graph is either claw-like or cycle-like.

□

Corollary 3.1.9 *Let r be a given row in a binary matrix M with three 1s per row. Then r is conflicting if and only if there exists a set of rows R of M such that $r \in R$ and the bipartite graph corresponding to R is either claw-like or cycle-like.*

Proof. This corollary follows directly from the previous theorem as in order for r to be conflicting, r must be in a minimal conflicting set R, and R is a minimal conflicting set if and only if its corresponding bipartite graph is either claw-like or cycle-like. □

Deciding whether $CI(r) > 0$ for a row r in a binary matrix M with three 1s per row can be done in polynomial time. This requires first checking whether r belongs to a claw-like graph or either of the first two cases of cycle-like graphs. Since all of these patterns have at most four row vertices, there are only a polynomial number of cases to check. Now if r does not belong to any of the claw-like or the first two cases of cycle-like graphs, we need to check whether r belongs to a chordless cycle in $B(M)$. We can use Dijkstra's algorithm for finding the shortest path between two neighbour rows of r in $B(M)$, and this can be done in polynomial time. However, computing $CI(r)$ is $\#W[1]$-complete [15] as it requires computing the number of chordless cycles containing r in $B(M)$; still it can be done using the principle of exclusion [44].

3.2 An Algorithm to Decide Whether a Row is Conflicting

We now consider the following problem: given a row of an $m \times n$ binary matrix M, does this row belong to at least one MCS? As far as we know, the complexity of this problem is still open. Our contribution is to show that, if each row of M has at most k 1s, where k a constant, then this question can be answered in time that is polynomial in n [8].

We first consider the case where the row in question does not belong to any of the minimal conflicting sets constructed from M_{II_n}, M_{III_n}, M_{IV}, M_V.

Definition 3.2.1 In a bipartite graph $B(M)$, we call a vertex representing a row of M a *row-vertex*, and a vertex representing a column of M a *column-vertex*. We denote the set of row-vertices and the set of column-vertices of $B(M)$ as $RV(B)$ and $CV(B)$ respectively.

Definition 3.2.2 Let r be a row-vertex in a bipartite graph B such that $B(J)$ is contained in B for some $J \in \{M_{II_n}, M_{III_n}, M_{IV}, M_V\}$. We say that r is $B(J)$-*conflicting* if $RV(B)$ is a minimal conflicting set.

Observe that in the previous definition, $RV(B)$ and $RV(B(J))$ must have the same cardinality in order for $R(B)$ to be a minimal conflicting set, whereas $CV(B)$ may have greater cardinality than $CV(B(J))$. For example, the set of row-vertices in each of the bipartite graphs that are claw-like or cycle-like is a minimal conflicting set, and $Claw_1, Claw_2, Claw_3$ all contain $B(M_{III_1})$, $Claw_4$ contains $B(M_{IV})$, and $Cycle_i$ contains $B(M_{I_n})$ for some n, for each $i \in \{1, 2, 3\}$.

Proposition 3.2.3 *Let M be a given binary matrix. For a given row $r_s \in R(M)$, suppose r_s is not $B(J)$-conflicting for $J \in \{M_{I_1}, M_{I_2}, M_{II_n}, M_{III_n}, M_{IV}, M_V\}$. Then r_s is conflicting if and only if there exists a subgraph B of $B(M)$ containing a cycle $C = (r_a c_{ab} r_b \dots r_x c_{xs} r_s c_{sy} r_y \dots r_z c_{za})$ such that $r_x \cap r_s \cap r_y = \emptyset$, where $c_{ij} \in r_i \cap r_j$.*

Proof.
($\Rightarrow$) Since r_s is conflicting and not $B(J)$-conflicting for $J \in \{M_{I_1}, M_{I_2}, M_{II_n}, M_{III_n}, M_{IV}, M_V\}$, r_s is $B(M_{I_n})$-conflicting, $n \geq 3$. Hence r_s belongs to some subgraph B which contains a cycle C such that $RV(B)$ is an MCS and $|RV(C)| = |RV(B)| \geq 5$.

Suppose to the contrary that for all subgraphs containing cycles of the form $(r_a c_{ab} r_b \dots r_x c_{xs} r_s c_{sy} r_y \dots r_z c_{za})$, we have $r_x \cap r_s \cap r_y \neq \emptyset$. Then there is some $c_{xsy} \in r_x \cap r_s \cap r_y$ on C. Now $B - \{r_s\}$ still contains a cycle, since r_x is still adjacent to r_y. So $B - \{r_s\}$ does not have the $C1P$, which contradicts that $RV(B)$ is a MCS.
($\Leftarrow$) We consider a minimal subgraph B containing a cycle $C = (r_a c_{ab} r_b \dots r_x c_{xs} r_s c_{sy} r_y \dots r_z c_{za})$ satisfying the condition:

$$r_x \cap r_s \cap r_y = \emptyset. \tag{3.1}$$

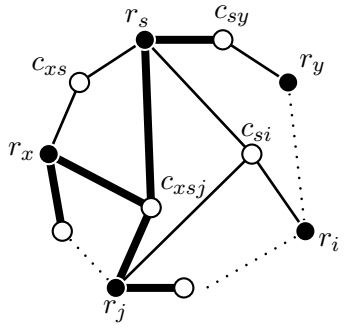
Then C is a minimal cycle satisfying this condition and $RV(B) = RV(C)$. Since r_s is not $B(J)$-conflicting for $J \in \{M_{I_1}, M_{I_2}\}$, $RV(B) \geq 5$. We prove that B is chordless, whence, by Proposition 3.1.6, $RV(B)$ is an MCS and r_s is conflicting.

Suppose toward a contradiction that B has one or more chords. We first observe that we cannot have chords in $B - [(r_x \cap r_s) \cup \{r_s\} \cup (r_s \cap r_y)]$. For, supposing we had a chord $b_i b_j$ in $B - [(r_x \cap r_s) \cup \{r_s\} \cup (r_s \cap r_y)]$, then

$$(b_i \dots r_x c_{xs} r_s c_{sy} r_y \dots b_j)$$

would be a smaller cycle satisfying condition (3.1), contradicting the minimality of C. Therefore any chord must have an endpoint in $(r_x \cap r_s) \cup \{r_s\} \cup (r_s \cap r_y)$, and so must be adjacent to r_s.

Let $(r_s c_{si} r_i)$ be a chord in B, then $i \neq x, y$. Since C satisfies condition (3.1), c_{si} is not adjacent to both r_x and r_y. Without loss of generality, we suppose c_{si} is not adjacent to r_x. Let r_j, $j \neq s$, be the row vertex adjacent to c_{si} such that none of the vertices on the cycle $C^* = (c_{si} r_j \dots r_x r_s)$ is adjacent to c_{si} except r_s and r_j. Since r_s is not M_{I_1}, M_{I_2} conflicting, $RV(C^*) \geq 5$. But C is a minimal cycle satisfying condition (3.1), so that C^* must not satisfy condition (3.1), and there must be some $c_{xsj} \in r_x \cap r_s \cap r_j$. But then r_s is B_{III_1}-conflicting, i.e. r_s belongs to a claw as illustrated in the graph below, a contradiction.



□

The following corollary follows immediately from this proposition.

Corollary 3.2.4 *Let M be a given binary matrix. For a given row $r_s \in R(M)$, if r_s is not $B(J)$-conflicting for $J \in \{M_{I_1}, M_{I_2}, M_{II_n}, M_{III_n}, M_{IV}, M_V\}$, and $B(M)$ contains a cycle $C = (r_a c_{ab} r_b \dots r_x c_{xs} r_s c_{sy} r_y \dots r_z c_{za})$ such that $r_i \cap r_s = \emptyset$ for all $i \neq x, s, y$ and $r_x \cap r_s \cap r_y \neq \emptyset$, then r_s is not conflicting.*

We next consider the case where a row may be $B(J)$-conflicting for $J \in \{M_{II_n}, M_{III_n}, M_{IV}, M_V\}$. These four Tucker's patterns $M_{II_n}, M_{III_n}, M_{IV}, M_V$ have the following crucial property, that follows from their definition: if no row of an $m \times n$ binary matrix M has more than k 1s, and a sub-matrix T of M contains a Tucker pattern $M_{II_n}, M_{III_n}, M_{IV}$ or M_V , then $3 \leq t \leq k+1$, where t is the number of rows of T. This proves the following result.

Proposition 3.2.5 *If no row of an $m \times n$ binary matrix M has more than k 1s, then deciding if a row r_s belongs to a sub-matrix T of M that contains a Tucker pattern $M_{II_n}, M_{III_n}, M_{IV}$ or M_V can be done in $O(k^2 m^{k+1}(n+m+c))$ time and $O(n+m)$ space, where c is the number of 1s in M.*

With Algorithm 3.1 we can decide whether a given row r_s of M is conflicting, where M is a binary matrix has at most k 1s for each row. The algorithm relies on the following subroutines and definitions:

1. Subroutine $CheckBTP$ checks (using a brute-force approach) whether the matrix formed by a collection q of rows is a MCS.

2. Subroutine $ExcPath$ checks for given rows (r_x, r_y) whether there is a path in $B(M)$ between r_x and r_y that excludes r_s.

3. $P(r_s)$ is the set of all pairs (r_x, r_y) of rows, such that $r_x \cap r_s \neq \emptyset \neq r_y \cap r_s$ and $r_x \cap r_y \cap r_s = \emptyset$. (Note that there are at most $O(m^2)$ such pairs of rows.)

4. $Q(r_s)$ is the collection of all sets of ρ rows, with $4 \leq \rho \leq k+1$, in which one row is r_s.

Algorithm 3.1 Conflicting Row

```
1: for each q ∈ Q(r_s)
2:     if CheckBTP(q) then return conflicting
3: for each (r_x, r_y) ∈ P(r_s)
4:     if ExcPath(r_x, r_y) then return conflicting
5: return nonconflicting
```

Theorem 3.2.6 *Let M be an $m \times n$ non-$C1P$ matrix with at most k 1s in each row, and let r_s be a row of M. Deciding whether r_s belongs to at least one MCS can be done in $O(k^2 m^{k+1}(n+m+c))$ time and $O(n+m)$ space, where c is the number of 1s in M.*

As far as we know, this is the first use of Tucker patterns in an efficient algorithm related to the MCS. However, the complexity status of deciding if a row belongs to an MCS when the rows of a matrix can have an arbitrary number of 1s is still open. In the next chapter, we present some approaches for computing all MCS of a given binary matrix.

Chapter 4

Computing All Minimal Conflicting Sets

In this chapter, we present three ideas to generate all the minimal conflicting sets of a given binary matrix M. We first introduce a naive idea [37], which involves deleting each row from a non-$C1P$ set and checking whether the resulting set has the $C1P$ or not. The second idea uses Boolean functions, and requires generating all the maximal $C1P$ sets at the same time. This work, originating from an idea of Tamon Stephen, was done in collaboration with him, Utz-Uwe Haus, and Cedric Chauve, and has been submitted [8] for publication. The third idea is a backtracking approach, which only gives us a way to generate all minimal conflicting sets in matrices with two 1s per row.

	Reference	Key features	Runtime
Alg 4.1 (naive)	Stoye & Wittler [37]	Easy to implement.	exponential
Alg 4.2 (monotone Boolean functions)	Fredman & Khachiyan [16]	Must also generate $MC1P$	quasi-polynomial in output
Backtracking	Read & Tarjan [36]	Must have two 1s per row.	polynomial in output

Table 4.1: Algorithms for Generating all MCS

4.1 Existing Algorithms

In this section, we present an existing algorithm, by Wittler and Stoye [37], for generating all minimal conflicting sets of a given binary matrix M. This is a naive algorithm which has exponential time complexity, and, as far as we know, it is the only previously proposed algorithm for computing all MCS. We begin by defining a subroutine, Algorithm 4.1.

Algorithm 4.1 Generate a MCS $R \subseteq R(M)$

```
1: Let M be a non-C1P binary matrix and R(M) = {r_1, r_2, ..., r_k}.
2: R ← R(M)
3: for i from 1 to k do
4:     if R − {r_i} is conflicting then R ← R − {r_i}
5: return R
```

Proposition 4.1.1 *The R returned in Algorithm 4.1 is an MCS.*

Proof. Let R^* be the return value of Algorithm 4.1. Since each time we delete an element r_i from R, we have $R - \{r_i\}$ conflicting, the final R^* must be conflicting. Hence R^* does not have the $C1P$. On the other hand, for each $r_j \in R^*$, $R^* - \{r_j\}$ must have the $C1P$, since otherwise there exists some $r_k \in R^*$ such that $R^* - \{r_k\}$ is conflicting, which is impossible by the definition of the algorithm. □

Example 4.1.2 Suppose $R = R(M) = \{r_1 = \{1,2\}, r_2 = \{2,3\}, r_3 = \{3,4\}, r_4 = \{4,5\}, r_5 = \{3,1\}$. Then in applying Algorithm 4.1, we will have:

i	$R - \{r_i\}$	$C1P$	action
1	$\{\{2,3\},\{3,4\},\{4,5\},\{3,1\}\}$	×	delete r_1 from R
2	$\{\{3,4\},\{4,5\},\{3,1\}\}$	✓	keep r_2 in R
3	$\{\{2,3\},\{4,5\},\{3,1\}\}$	✓	keep r_3 in R
4	$\{\{2,3\},\{3,4\},\{3,1\}\}$	×	delete r_4 from R
5	$\{\{2,3\},\{3,4\}\}$	✓	keep r_5 in R

Therefore finally we return $R = \{r_2, r_3, r_5\}$, and it is easy to check that this is an MCS.

We give the algorithm of Wittler and Stoye as Algorithm 4.2. At the generic step of the algorithm, we will have a set $S = \{S_1, S_2, \ldots, S_\ell\}$ of MCS, and we will use an iterator, COMBO, which works as follows. The first time COMBO is called, it chooses one row r_i

from each S_i, and it returns the set $C = \{r_1, r_2, \ldots, r_\ell\}$. Each time it is called subsequently, it returns the set C generated by choosing a different combination of rows r_i. After it has exhausted all possible choices, it returns $FAIL$. If the set S is enlarged by adding a new element $S_{\ell+1}$, then COMBO resets.

Algorithm 4.2 Generate all MCS $R \subseteq R(M)$

```
1: Let S_1 be an MCS generated by Algorithm 4.1 on rows R(M).
2: S ← {S_1}
3: ℓ ← 1
4: C ← COMBO(S)
5: while C ≠ FAIL do
6:     R ← R(M)\C
7:     if R has the C1P then
8:         C ← COMBO(S)
9:     else
10:        ℓ ← ℓ + 1
11:        Let S_ℓ be an MCS generated by Algorithm 4.1 on rows R.
12:        S ← S ∪ {S_ℓ}
13:        C ← COMBO(S)
14: return S
```

Since for two different MCS S_i and S_j, we have $S_i \not\subset S_j$ and $S_j \not\subset S_i$, this algorithm never generates the same MCS twice, and guarantees that we generate all MCS of $R(M)$. This algorithm can require time $O(n^\ell)$ to terminate as it must check all combinations of rows of current known MCS.

In next section, we present another way to generate all MCS, using a monotone Boolean function approach, which is more efficient than this naive algorithm.

4.2 A Monotone Boolean Function Approach

In this section we present a way to generate all the MCS of a given matrix M by a monotone Boolean function approach. This method also generates all the complements of the maximal $C1P$ sets, while generating all the MCS, so it is called a joint generation algorithm.

Definition 4.2.1 A *Boolean function* is a mapping $f : \{0,1\}^m \to \{0,1\}$. Each $x = (x_1 x_2 \ldots x_m) \in \{0,1\}^m$ is called a *Boolean vector*, and $x_1, x_2, \ldots, x_m$ are called *Boolean variables*. A Boolean function f is called *monotone* if it satisfies the condition that $x \geq y$

implies $f(x) \geq f(y)$ for all $x, y \in \{0,1\}^m$.

Definition 4.2.2 Boolean variables $x_1, x_2, ..., x_m$ and their complements $\overline{x_1}, \overline{x_2}, \ldots, \overline{x_m}$ are called *literals*. A *clause* is a disjunction of literals which does not contain both x_i and $\overline{x_i}$. Similarly, a *term* is a conjunction of literals which does not contain both of x_i and $\overline{x_i}$.

Definition 4.2.3 We say a clause c is an *implicate* of a Boolean function f if $f \Rightarrow c$. If there does not exist a clause $c^* \neq c$ such that $f \Rightarrow c^* \Rightarrow c$, then we say c is a *prime implicate* of f.

Definition 4.2.4 We define a term d to be an *implicant* of a Boolean function f if $d \Rightarrow f$. If there does not exist a term $d^* \neq d$ such that $d \Rightarrow d^* \Rightarrow f$, then we say d is a *prime implicant* of f.

Definition 4.2.5 The irredundant conjunctive normal form(CNF) and disjunctive normal form(DNF) of a Boolean function f are defined respectively as:

$$CNF = \bigwedge_{I \in C} \bigvee_{i \in I} x_i \qquad DNF = \bigvee_{J \in D} \bigwedge_{j \in J} x_j$$

where C and D are the set of prime implicates and the set of prime implicants of f respectively.

If $C' \subset C$ and $D' \subset D$, we denote the CNF and DNF of f on C' and D' respectively by:

$$CNF[C'] = \bigwedge_{I \in C'} \bigvee_{i \in I} x_i \qquad DNF[D'] = \bigvee_{J \in D'} \bigwedge_{i \in J} x_j$$

Remark 4.2.6 By definition, we can see that $CNF(x) = f(x) = DNF(x)$ for all $x \in \{0,1\}^m$.

Definition 4.2.7 Let $r_1, r_2, \ldots, r_m$ be the m rows of a given binary matrix M. For each set $S \subseteq \{r_1, r_2, \ldots, r_m\}$ of rows, we define $x(S) = (x_1 x_2 \ldots x_m)$ to be the Boolean vector such that $x_i = 1$ if and only if $r_i \in S$ and $\overline{x}(S) = (\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_m)$ to be the Boolean vector such that $\overline{x}_i = 0$ if and only if $r_i \in S$.

Definition 4.2.8 For each binary matrix M, we define a pair of Boolean functions as follows, where S varies over the subsets of $R(M)$:

$$f_M(x(S)) = \begin{cases} 0 & \text{if } S \text{ has the } C1P \\ 1 & \text{if } S \text{ does not have the } C1P \end{cases}$$

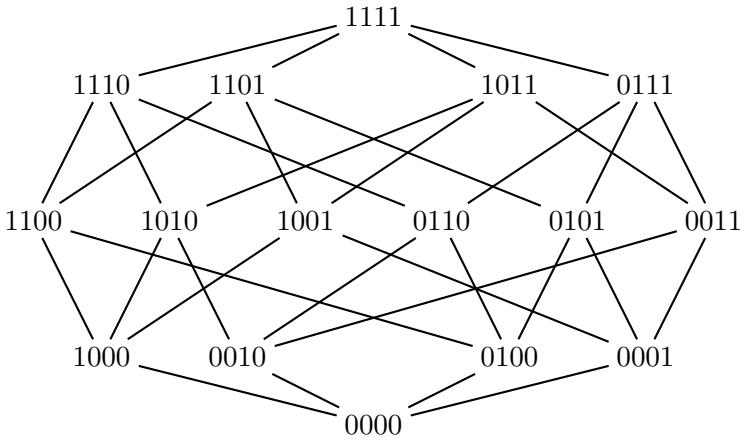
$$h_M(\overline{x}(S)) = \begin{cases} 0 & \text{if } S \text{ does not have the } C1P \\ 1 & \text{if } S \text{ has the } C1P. \end{cases}$$

We see that f_M is monotone, since, for any $x, y \in \{0,1\}^m$, $x \geq y$ means the set S_y of rows represented by y is a subset of the set S_x of rows represented by x. If S_y has the $C1P$, $0 = f_M(y) \leq f_M(x)$. If S_y does not have the $C1P$, then $S_y \subseteq S_x$ implies S_x has at least all the rows of S_y, and therefore S_x cannot have the $C1P$, so $f_M(y) = 1 \leq 1 = f_M(x)$.

Example 4.2.9 Consider the following binary matrix M:

	c_1	c_2	c_3	c_4
r_1	**1**	**1**	0	**1**
r_2	0	**1**	**1**	0
r_3	**1**	0	**1**	0
r_4	0	**1**	**1**	**1**

One can check that $S_1 = \{r_1, r_2, r_3\}$ and $S_2 = \{r_1, r_3, r_4\}$ are the only minimal conflicting sets, whence $f_M(x(S)) = 1$ if and only if $x(S) \in \{1110, 1011, 1111\}$. Considering the lattice below, in which points are connected if and only if they differ in exactly one place,



we can see that if $f_M(x) = 1$ for some lattice point x, and if for every lattice point y on some path below x we have $f_M(y) = 0$, then x represents an MCS. Since every MCS is contained

in one or more conflicting sets, the set of MCS is indeed the set of prime implicants in the DNF of f_M.

We observe in this example that $h_M(\overline{x}(S)) = 0$ if and only if $\overline{x}(S) \in \{\overline{1110}, \overline{1011}, \overline{1111}\} = \{0001, 0100, 0000\}$, so that we have $h_M(\overline{x}) = \overline{f_M}(x)$ for $x \in \{1110, 1011, 1111\}$.

Definition 4.2.10 The dual of a function f is defined as $f^d(x) = \overline{f}(\overline{x})$, where $\overline{f}$ and $\overline{x}$ are the complement of f and x respectively.

Proposition 4.2.11 *The functions f_M and h_M from Definition 4.2.8 are mutually dual.*

The proof for this proposition is trivial since by definition we have $\overline{f_M}(x) = h_M(\overline{x})$, which implies $f_M(x) = \overline{h_M}(\overline{x})$ and $\overline{f_M}(\overline{x}) = h_M(x)$.

Note that since f_M is a monotone function, all prime implicates and prime implicants of f_M are monotone; moreover, since $h_M = f_M^d$, h_M is also monotone [14].

Let F_M and H_M be the set of prime implicants of f_M and h_M respectively, and consider the DNF of f_M and h_M:

$$f_M = \bigvee_{I \in F_M} \bigwedge_{i \in I} x_i \qquad h_M = \bigvee_{J \in H_M} \bigwedge_{j \in J} x_j.$$

We know that for any Boolean function f with prime implicates C and prime implicants D, there are bijections $C \rightleftharpoons Max\{x : f(x) = 0\}$ and $D \rightleftharpoons Min\{x : f(x) = 1\}$ [21]. Therefore we have:

$F_M \rightleftharpoons Min\{x(S) : f_M(x(S)) = 1\} \rightleftharpoons Min\{x(S) : S \text{ is conflicting}\} \rightleftharpoons \{\text{MCS}\}$

$H_M \rightleftharpoons Min\{\overline{x}(S) : h_M(\overline{x}(S)) = 1\} \rightleftharpoons Min\{\overline{x}(S) : S \text{ has the } C1P\}$

$\rightleftharpoons \overline{Max\{x(S) : S \text{ has the } C1P\}} \rightleftharpoons \{\overline{MC1P}\}$,

where $MC1P$ denotes a subset of the rows of a binary matrix that has the $C1P$ and is maximal in that respect, and $\overline{MC1P}$ denotes the complement of such a set of rows. We also denote the set of all $MC1P$ ($\overline{MC1P}$) by $MC1P$ ($\overline{MC1P}$).

We can write the DNF of f_M and h_M as:

$$f_M = \bigvee_{I \in MCS} \bigwedge_{i \in I} x_i \qquad h_M = \bigvee_{J \in \overline{MC1P}} \bigwedge_{j \in J} x_j$$

We now consider the *Dual Problem*:

For a given pair of irredundant DNFs

$$DNF[C'] = \bigvee_{I \in C'} \bigwedge_{i \in I} x_i \qquad DNF[D'] = \bigvee_{J \in D'} \bigwedge_{j \in J} x_j,$$

check if they satisfy Condition (4.1) (in which case they are dual); otherwise find a Boolean vector x^* satisfying Condition (4.2) (thereby certifying that the DNFs are not dual).

$$DNF[C'](x_1, x_2, ..., x_m) = \overline{DNF[B]}(\overline{x_1}, \overline{x_2}, ..., \overline{x_m}) \text{ for all } x = (x_1, x_2, ..., x_m) \in \{0,1\}^m. \tag{4.1}$$

$$DNF[C'](\overline{x_1^*}, \overline{x_2^*}, ..., \overline{x_m^*}) = DNF[D'](x_1^*, x_2^*, ..., x_m^*) \tag{4.2}$$

Lemma 4.2.12 *Any dual disjunctive normal forms $DNF[C']$ and $DNF[D']$ must satisfy the condition:*

$$I \cap J \neq \emptyset, \qquad \textit{for all} I \in C' \quad \textit{and} \quad \textit{for all} J \in D'. \tag{4.3}$$

Proof. Suppose to the contrary that there exist $I \in C'$ and $J \in D'$ such that $I \cap J = \emptyset$. Let $x(J) = (x_1, x_2, ..., x_m)$ be the characteristic vector of J. Since $I \subseteq \overline{J}$, $x(J)$ will satisfy Condition (4.2), which contradicts the duality of $DNF[C']$ and $DNF[D']$. □

It is not hard to see that f_M and h_M satisfy Condition (4.3), i.e. any MCS intersects any $\overline{MC1P}$, since for any row that does not belong to a $MC1P$, it will belong to some MCS.

Let $H' \subseteq H = \{\overline{MC1P}\}$ and $F' \subseteq F = \{\text{MCS}\}$. Then $CNF[H'](x) \geq CNF[H](x)$ and $DNF[F](x) \geq DNF[F'](x)$. Along with Remark (4.2.6), we will have:

$$CNF[H'](x) \geq CNF[H](x) = CNF(x) = f(x) = DNF(x) = DNF[F](x) \geq DNF[F'](x). \tag{4.4}$$

Therefore $(H', F') = (H, F)$ if and only if $CNF[H'](x) = DNF[F'](x)$. Moreover, we have:

$$CNF[H'](x) = \bigwedge_{I \in H'} \bigvee_{i \in I} x_i$$

which implies

$$\overline{CNF}[H'](x) = \bigvee_{I \in H'} \bigwedge_{i \in I} \overline{x_i}$$

and

$$CNF^d[H'](x) = \overline{[CNF}[H'](\overline{x}) = \bigvee_{I \in H'} \bigwedge_{i \in I} x_i = DNF[H'].$$

Therefore $(H', F') = (H, F)$ if and only if $DNF[H'](x)$ and $DNF[F'](x)$ are mutually dual.

Corollary 4.2.13 *[21] If $DNF[C']$ and $DNF[D']$ satisfy Condition (4.3), then the Dual Problem can be solved in time $T = O(m^2) + (|C'| + |D'|)^{o(log(|C'|+|D'|))}$.*

By the previous corollary, we have a way to check whether $DNF[F']$ and $DNF[H']$ are mutually dual. If they are dual, then we learn that $(F', H') = (F, H)$ and we get all the MCS. If they are not dual, then instead we get a Boolean vector x^* which satisfies Condition (4.2). Now $x^* \notin F'$ implies $DNF[F'](x^*) = 0$, so that $DNF[H'](\overline{x^*}) = 0$, and $CNF[H'](x^*) = \overline{DNF}[F'](\overline{x^*}) = 1$.

Now evaluating $f_M(x^*)$ we split into two possible cases:

1. $f_M(x^*) = 0$. Then the set of rows represented by x^* has the $C1P$, but is not maximal. But $x^* \notin C'$ implies there is some $y^* \in Max\{x : f_M(x) = 0\}$ such that $x^* < y^*$. Now $I = \{i : y_i^* = 0\} \in C \backslash C'$, and therefore we can obtain a new prime implicate of f_M.

2. $f_M(x^*) = 1$. Then the set of rows represented by x^* is conflicting, but not minimal. Then we can find a $y^* \in Min\{x : f_M(x) = 1\}$ such that $y^* < x^*$. Now the set $J = \{j : y_j^* = 1\} \in D \backslash D'$, therefore we obtained a new prime implicant of f_M.

Thus we have the following theorem:

Theorem 4.2.14 *[16] Let $f : \{0,1\}^m \to \{0,1\}$ be a monotone Boolean function whose value at any point $x \in \{0,1\}^m$ can be determined in time t, and let C and D be the sets of the prime implicates and prime implicants of f respectively. Given two subsets $C' \subset C$ and $D' \subset D$ of total size $|C'| + |D'| < |C| + |D|$, a new element in $(C \backslash C') \cup (D \backslash D')$ can be found in time $O(m(t + m) + (|C'| + |D'|)^{o(log(|C'|+|D'|))})$.*

In our case, t is the time to test whether a given set has the $C1P$ or not, and this can be done in linear time. We start with $F' = \emptyset = H'$, and then in each iteration we generate a new element either in $F \backslash F'$ or in $H \backslash H'$, and repeat this process until we get the whole sets $F = \{\text{MCS}\}$ and $H = \{\overline{MC1P}\}$. We call this process $Gen(F, H, F', H')$.

Proposition 4.2.15 *[5] Gen(F,H,F',H') can be completed in time $m(poly(|F'| + |H'|) + O(m)) + T$.*

Thus, for any given binary matrix M, we can use the Boolean approach to generate all the minimal conflicting sets. This is a joint generation method, which generates all the complements of the $MC1P$ sets at the same time. It is an improvement over the algorithm of Stoye and Wittler described in previous section, which, after finding the complete list S of MCS, will continue to check whether the $MC1P$ as candidate conflicting sets before terminating. Moreover, as this does not keep the $MC1P$ sets explicitly, but instead uses backtracking, it may generate the same candidates repeatedly.

4.3 An Efficient Backtracking Approach for Matrices with Two 1s per Row

It appears clearly from the previous section that the monotone Boolean approach to generate all MCS suffers from a major problem that lies in the need to enumerate also all $MC1P$ sets. We have seen in Section 1.3.2 that when the considered binary matrix M has only two 1s per row, enumerating all MCS follows easily from enumerating all claws (that are defined by exactly three rows of M and can then be computed in polynomial time) and all cycles of the graph $B(M)$. We describe here an algorithm of Read and Tarjan for enumerating all cycles of a graph that has a time complexity that is polynomial in the number of cycles. This gives then an algorithm that enumerates all MCS in time polynomial with the number of MCS.

Enumerating all cycles of a given graph is a classic algorithmic problem, for which four well-known methods have been introduced. The first is an algorithm by Cartwright and Gleason [7] which uses the edge-digraph of a digraph. The second approach is based on the adjacency matrix [35, 45, 1]. The two remaining methods, the vector space approach [34, 24, 30, 43, 20] and the backtracking approach [36, 27, 29, 39, 13], are more generally used. The backtracking approach is the only one that guarantees a time complexity that is polynomial in the number of cycles of the considered graph.

Theorem 4.3.1 *[36] Let G be a graph with v vertices, e edges and c cycles. It is possible to enumerate all cycles of G in worst-case time $O(v + e + ec)$.*

Corollary 4.3.2 *Let M be an $m \times n$ binary matrix such that each row has exactly two entries 1. If M has p MCS, it is possible to enumerate all the MCS of M in worst-case time $O(m^3 + mp)$.*

Proof. The $O(m^3)$ term corresponds to enumerating all triplets of rows of M that have a common 1. The $O(mp)$ term follows from the $O(ec)$ term in Theorem 4.3.1 when applied to $B(M)$ that has exactly m edges. □

We now present the backtracking algorithm by Read and Tarjan [36] for enumerating all cycles of a given graph, which is considered to be simpler and theoretically more efficient than others.

Given a graph G with v vertices, the original idea of the backtracking process is introduced by Tiernan [40] and the procedure for listing all cycles is as follows:

We first number the vertices from 1 to v (arbitrarily). Then, starting with vertex number 1, we pick an edge that is adjacent to this starting vertex, thus beginning to build a path from the starting vertex. Next we pick an edge adjacent to the end vertex of the current path, to extend the current path. During this extending process, we have the restrictions that the same vertex cannot appear twice on the path, and that we cannot have vertices with smaller labels than the starting vertex. Each time we extend the current path, we check whether the end vertex on this extended path is adjacent to the starting vertex. If so, we output a cycle, and we keep trying all the other possibilities to extend the path. After trying all possible ways of extending the path, we back up and delete the last edge and try another possible edge as an extension. We repeat this process with each vertex as the starting vertex, and in this way we generate all cycles of the graph.

This procedure may be inefficient as many of the path extensions may not lead to a cycle. Read and Tarjan [36] improve this backtracking procedure by giving a way to select the starting vertex for each cycle and a way to put restrictions on the backtracking. They begin as follows:

1. Use depth-first search to number the vertices from 1 to v, and produce a spanning tree.

2. Accordingly, divide the edges of G into four sets:

   (a) The spanning tree.

   (b) The set of all *cycle-arcs*: edges going from descendants to ancestors in the tree.

   (c) The set of all *forward-arcs*: edges going from ancestors to descendants in the tree.

(d) The set of all *cross-arcs*: edges joining unrelated vertices in the spanning tree, i.e. edges that go from one subtree to another subtree.

We use depth-first search repeatedly to partition G into a set of trees and non-tree arcs, and this takes time $O(v + e)$. From another paper of Tarjan [39], we know that tree edges and forward-arcs always go from vertices with smaller labels to vertices with larger labels, while cycle-arcs and cross-arcs always go, vice-versa, from larger to smaller vertex labels. Moreover, each cycle-arc must be contained in at least one cycle and each cycle must end with a cycle-arc. Hence we can take the set of starting vertices to be exactly the set of vertices with cycle-arcs entering into them. This is more efficient than trying each vertex as a starting vertex.

3. For a starting vertex s, we follow the recursive backtracking procedure presented below, to build up a path which may lead to a cycle:

   (a) Let P be the current path and u be the last vertex on P.

   (b) Search for a vertex w such that (u, w) is an edge and there is a path from w to s avoiding P except at s.

   (c) Use a search to determine how far $P + w$ can be extended uniquely toward s, i.e. extend $P + w$ until there are two possible choices for the next extending vertex.

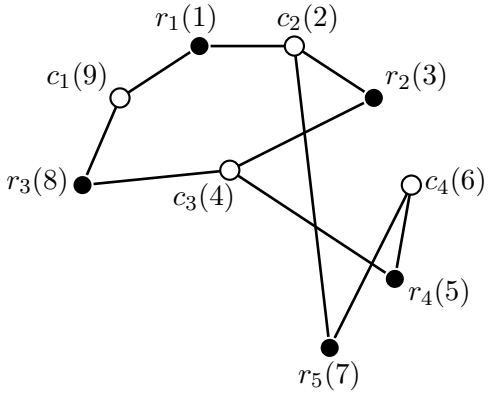
   (d) Apply the backtracking procedure recursively if we have two choices for the next extending vertex.

   We carry out the above process for each starting vertex in order to generate all cycles. Since in this process we do not apply backtracking recursion until two possible choices are available, this puts a restriction on the backtracking, improving the efficiency of the procedure.

Example 4.3.3 We use the backtracking method to generate all MCS of the binary matrix M:

	c_1	c_2	c_3	c_4
r_1	**1**	**1**	0	0
r_2	0	**1**	**1**	0
r_3	**1**	0	**1**	0
r_4	0	0	**1**	**1**
r_5	0	**1**	0	**1**

by computing the cycles in $B(M)$, which is illustrated below. We suppose the labels $1, 2, \ldots, 9$ in round brackets were generated by a depth-first search.

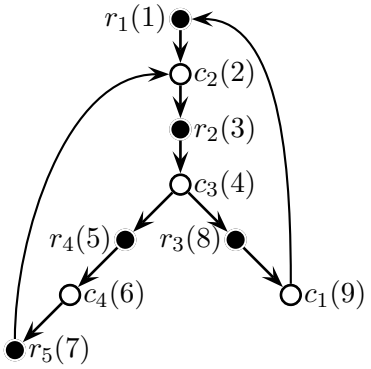


We can get a directed graph from $B(M)$ by replacing each edge by two directed edges having opposite directions. We pick the directed edges in the following way:

1. Let the tree edges be the directed edges selected by following the order of the depth-first search.

2. For the remaining edges, we choose the directed edges that go from larger label to smaller label.

By applying the above rule, we get the directed graph corresponding to $B(M)$ as:



We then apply the backtracking procedure. Since the only cycle-arcs are edges $(9, 1)$ and $(7, 2)$, the only starting vertices are 1 and 2. Starting from 1, we construct the path (1234), and then there are two possibilities for the next vertex. Since there is no path from 5 to 1, we reject 5 as the next vertex. There remains then only one way to extend the path, and we compute the cycle (1234891).

Next we consider the other possible starting vertex, 2. We first construct the path (234), and then have two possibilities for the next vertex, 5 and 8. There are paths from both vertices back to 2, but the path that involves 8 must have 1 on it, and 1 is a smaller label than 2, the starting vertex, so we must reject 8 as the next vertex. We then have only one way to extend the path, and we get the cycle (2345672).

At this point the procedure halts, and we have found that the only cycles in $B(M)$ are (1234891) and (2345672). These correspond to the MCS $\{r_1, r_2, r_3\}$ and $\{r_2, r_4, r_5\}$ respectively.

4.4 Experimental Results

We present in this last section some preliminary results on synthetic data (generated matrices) with two 1 per row. To compute MCS, we used the implementation of the joint generation method which is publicly available [23] with an oracle to test the $C1P$ based on the algorithm described in [31].

We generated 10 datasets of adjacencies (the rows of the binary matrices each contain two 1s) with $n = 40$ and $m = 45$. Each dataset contains exactly 39 true positive (rows $\{i, i + 1\}$ for $i = 1, \ldots, 39$) and 6 random false positives (rows $\{i, j\}$ with $j > i + 1$). These parameters were chosen to simulate moderately large datasets that resemble real datasets. For a given dataset, the *conflicting ratio* (CR) of a row is the ratio between the CI of this row and the number of MCS. Similarly, the *MC1P ratio* (MR) of a row is the ratio between the number of $MC1P$ that contain the row and the total number of $MC1P$. The MCS *rank* of a row is its ranking (between 1 and 45) when rows are ranked by increasing CR. The $MC1P$ *rank* of a row is its ranking when rows are ranked by increasing MR. Table 4.2 shows some statistics on these experiments. All experiments took at most a few minutes to complete.

First, we can notice the large difference between the number of MCS and the number of $MC1P$. This shows that most computation time, in the joint generation, is spent generating

Dataset	Number of MCS	Number of $MC1P$	Average FP_CR	Average TP_CR	Average FP_MR	Average TP_MR	Average FP MCS rank	Average FP $MC1P$ rank
1	55	8379	0.41	0.37	0.34	0.77	18.83	6.83
2	43	4761	0.36	0.32	0.3	0.84	20.33	6
3	38	9917	0.4	0.22	0.34	0.79	33.17	7
4	46	4435	0.5	0.35	0.41	0.8	33.33	9
5	59	6209	0.44	0.3	0.36	0.76	28.33	6
6	45	13791	0,47	0.2	0.39	0.8	32.67	4.67
7	61	2644	0.44	0.31	0.37	0.8	28.83	5.83
8	50	3783	0.43	0.28	0.36	0.81	34.5	6.83
9	57	2575	0.51	0.37	0.43	0.81	32.83	5.17
10	60	3641	0.45	0.31	0.38	0.83	26.33	7.83

Table 4.2: Statistics on MCS and $MC1P$ on simulated adjacencies datasets. FP_CR is the Conflicting Ratio for False Positives, TP_CR is for CR the True Positives, FP_MR is the $MC1P$ ratio for False Positives and TP_MR is the MR for True Positives.

Dataset	$[0,.1]$	$(.1,.2]$	$(.2,.3]$	$(.3,.4]$	$(.4,.5]$	$(.5,.6]$	$(.6,.7]$	$(.7,.8]$	$(.8,.9]$	$(.9,1]$
MCS ALL	52	16	75	205	87	14	1	0	0	0
MCS FP	0	0	14	31	13	2	0	0	0	0
MCS TP	52	16	61	174	74	12	1	0	0	0
$MC1P$ ALL	10	2	7	18	32	50	73	44	20	194
$MC1P$ FP	0	0	0	6	22	44	64	40	20	194
$MC1P$ TP	10	2	7	12	10	6	9	4	0	0

Table 4.3: Distribution of the MCS and $MC1P$ ratios for all rows (ALL), false positives (FP) and true positives (TP). Each cell of the table contains the number of rows whose ratio is in the interval for the column.

$MC1P$. However if, as expected, false positives have, on average, a higher conflicting ratio than true positives, and conversely a lower $MC1P$ ratio than true positives, it is interesting that the $MC1P$ ratio discriminates much better between false positives and true positives than the conflicting ratio. This is seen in the MCS and $MC1P$ ranks: the false positives have an average MCS rank of 28.91, well below the rank that would be expected if they were the rows that have the highest CI (42.17), while they have an average $MC1P$ rank of 6.52, quite close of the 3.5 rank expected if they belonged to the fewest $MC1P$. To get a better understanding of the usefulness of the MCS ratio and $MC1P$ ratio, Table 4.3 shows the rough distribution of these ratios.

These results suggest that the increased computation required by generating $MC1P$

brings valuable information in discriminating false positives from true positives, and that the $MC1P$ ratio is a better information to rank rows when trying to compute a maximal $MC1P$ subset of rows. However, more experiments are needed in order to extend these computations to more general matrices.

Chapter 5

Conclusion and Perspectives

The main contribution of this thesis is twofold: a better understanding of the combinatorial nature of minimal conflicting sets for the consecutive ones property, and algorithms to study conflicting sets.

As far as we know, our algorithm based on Tucker's patterns to decide if a row of a binary matrix belongs to at least one MCS is the first to use these patterns efficiently, in solving a problem related to the $C1P$. However, our work relies on the assumption that the matrices have a bounded number of 1s in each row, and the complexity status of deciding whether a row belongs to an MCS when the number of 1s in each row is arbitrary, is still open.

We also described a very general approach for enumerating all MCS of a binary matrix based on the joint-generation algorithm for monotone Boolean functions. This problem is hard in general, but it remains to be seen if the monotone Boolean functions related to conflicting sets have some properties that can make it more tractable. In any case, our experimental results suggest that the number of MCS and $MC1P$ makes this approach impractical for large binary matrices.

However, we also showed that in the cases of matrices with two 1s per row, cycle enumeration with a backtracking algorithm offers an efficient alternative to enumerate all MCS. There are two ways to extend the efficient approach (at least with respect to the number of MCS of a matrix M) described above. The first one consists in enumerating subgraphs defining MCS in the bipartite graph $B(M)$. However, this approach requires that we have a characterization of what is an MCS as a subgraph of $B(M)$, and our results on matrices with three 1s per row suggest this seems to be a hard problem. The best approach we can think about right now would consist in enumerating all Tucker patterns and then checking

for each if it is an MCS or not. Dom [11] gives algorithms for checking all Tucker patterns in polynomial time. A better understanding of the relationship between Tucker patterns and MCS could make this general approach efficient.

Another way to generalize the principle of the backtracking approach would deal directly with the rows of a matrix. One could think about a way of ordering these rows and listing them (as the edges of the graph when listing all cycles of a graph) until the set of rows did not have the $C1P$, in which case it would be tested for being an MCS or not. In both cases (MCS or not), it is not necessary to extend this set, as any other set of rows that contains a non-$C1P$ subset is not an MCS. An issue here would then be to test efficiently if a set of rows that does not have the $C1P$ is an MCS as the naive approach that removes each row and tests for the $C1P$ is relatively costly (although still tractable). Partition refinement could be a way to improve the efficiency of this test in the dynamic context of listing subsets of rows by backtracking.

A natural idea when dealing with hard counting problems such as computing the conflicting index of a row of a binary matrix is random generation. In the context of MCS the idea is then to generate such structures uniformly and at random. However, it is known that both generating cycles in a graph randomly and uniformly, and generating minimal true clauses for monotone Boolean functions, are hard problems [25] and there is little hope of this being tractable for MCS.

From a more applied point of view, it is interesting to remark that, for matrices with exactly two 1s in each row, true positive rows define a set of paths in the graph $B(M)$, representing ancestral genome segments, while in false positive rows $\{i, j\}$ – unless i or j is an extremity of such a path (in which case it does not exhibit any combinatorial sign of being a false positive) – both the vertices i and j belong to a claw in $B(M)$. And it is easy to detect all edges in this graph with both ends belonging to a claw. In order to extend this approach to more general datasets, where not every row of a binary matrix M contains exactly two 1s, it would be helpful to understand better the impact of adding a false positive row in M. The most promising approach would be to start from the partition refinement obtained from all true positive rows and form a better understanding of the combinatorial structure of connected components of the overlap graph that do not have the $C1P$.

Finally, our experiments also suggest that $MC1P$ sets provide valuable information for detecting false positives in binary matrices, and it would then be interesting to attack questions such as enumerating or sampling these structures.

Bibliography

[1] M. T. Ardon, N. R. Malik. A Recursive Algorithm for Generating Circuits and Related Subgraphs. *5th Asilomar Conference on Circuits and Systems*, pp. 279-284. 1971.

[2] J. C. Boland, C. G. Lekkerkerker. Representation of a Finite Graph by a Set of Intervals on the Real Line. *Fundamenta Mathematicae*, 51, pp. 45-64. 1962.

[3] K.S. Booth. PQ-tree Algorithms. *Ph.D. Thesis, University of California, Berkeley.* 1975.

[4] K.S. Booth, G.S. Lueker. Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-tree Algorithms. *Journal of Computer and System Sciences*, 13, pp. 335-379. 1976.

[5] E. Boros, K. Elbassioni, V. Gurvich, L. Khachiyan. Dual-Bounded Hypergraphs: A Survey. *Proceedings of the SIAM Workshop on Discrete Mathematics and Data Mining*, pp. 87-98. 2002.

[6] F. Boyer, C. Chauve, A. McPherson, A. Ouangraoua, E. Tannier. Prediction of Contiguous Ancestral Regions in the Amniote Ancestral Genome. *Proceedings of the International Symposium on Bioinformatics Research and Applications*, LNB, 5542, pp. 173-185. 2009.

[7] D. Cartwright, T. C. Gleason. The Number of Path and Cycles in a Digraph. *Psychometrika*, 31, pp. 179-199. 1966.

[8] C. Chauve, U.U. Haus, T. Stephen, V.P. You. Minimal Conflicting Sets for the Consecutive Ones Property in Ancestral Genome Reconstruction. Submitted.

[9] C. Chauve, J. Manuch, M. Patterson. On the Gapped Consecutive-Ones Property. *Proceedings of EuroComb 2009, Electronic Notes in Discrete Mathematics*. To Appear.

[10] C. Chauve and E. Tannier. A Methodological Framework for the Reconstruction of Contiguous Regions of Ancestral Genomes and Its Application to Mammalian Genomes *PLoS Computational Biology*, 4, e1000234. 2008.

[11] M. Dom. Recognition, Generation, and Application of Binary Matrices with the Consecutive-Ones Property. Dissertation, Institut für Informatik, Friedrich-Schiller-Universität Jena, Germany. 2008.

[12] M. Dom, J. Guo, and R. Niedermeier. Approximability and Parameterized Complexity of Consecutive ones Sub-matrix Problems. *Proceedings of the 4th Annual Conference on Theory and Applications of Models of Computation*, LNCS, 4484, pp. 680-691. 2007.

[13] A. Ehrenfeucht, L. D. Fosdick, L. J. Osterweil. An Algorithm for Finding the Elementary Circuits of a Directed Graph. *Tech. Rep. CU-CS-024-73, Department of Computer Science*, University of Colorado, Boulder. 1973.

[14] T. Eiter, K. Makino, G. Gottlob. Computational Aspects of Monotone Dualization: A Brief Survey. *Discrete Applied Mathematics*, 156, pp. 2035-2049. 2008.

[15] J. Flum, M. Grohe. The Parameterized Complexity of Counting Problems. *The 43rd Annual IEEE Symposium on Foundations of Computer Science*, pp.538. 2002.

[16] M.L. Fredman, L. Khachiyan. On the Complexity of Dualization of Monotone Disjunctive Normal Forms. *Journal of Algorithms*, 21, pp. 618628. 1996.

[17] D. R. Fulkerson, O. A. Gross. Incidence Matrices and Interval Graphs. *Pacific Journal of Mathematics*, 18, pp. 835-855. 1965.

[18] Y. Ganjali, M. T. Hajiaghayi. A Note on the Consecutive ones Sub-matrix Problem. *Information Processing Letters*, 83, pp. 163-166. 2001.

[19] S. P. Ghosh. File Organization: The Consecutive Retrieval Property. *Communications of the ACM*, 15, pp. 802-808. 1972.

[20] N. E. Gibbs. A Cycle Generation Algorithm for Finite Undirected Linear Graphs. *Journal of the ACM*, 16, pp. 564-568. 1969.

[21] V. Gurvich, L.Khachiyan. On Generating the Irredundant Conjunctive and Disjunctive Normal Forms of Monotone Boolean Functions. *Discrete Applied Mathematics*, 96-97, pp. 363-373. 1999.

[22] M. Habib, R. McConnell, C. Paul, L. Viennot. Lex-BFS and Partition Refinement, with Applications to Transitive Orientation, Interval Graph Recognition and Consecutive Ones Testing. *Theoretical Computer Science*, 234, pp. 59-84. 2000.

[23] U.U. Haus and T. Stephen. `CL-JOINTGEN`: A Common Lisp Implementation of the Joint Generation Method. Available at `http://primaldual.de/cl-jointgen/`. 2008.

[24] P. A. Honkanen, H. T. Hsu. A Fast Minimal Storage Algorithm for Determining All the Elementary Cycles of a Graph. *Computer Science Department*, Pennsylvania State University, University Park. 1972.

[25] M. Jerrum, L.G. Valiant, V.V. Vazirani. Random Generation of Combinatorial Structures from a Uniform Distribution. *Theoretical Computer Science*, 43, pp. 169-188. 1986.

[26] Q. Ji, Z. Luo, C. Yuan, J. Wible, J. Zhang, J. Georgi. The Earliest Known Eutherian Mammal. *Nature*, 416, pp. 816-822. 2002.

[27] D. B. Johnson. Finding All the Elementary Circuits of a Directed Graph. *SIAM Journal of Computing*, 4, pp. 77-84. 1975.

[28] D. G. Kendall. Some Problems and Methods in Statistical Archaeology. *World Archaeology*, 1, pp.68-76. 1969.

[29] P. E. Lauer, J. L. Szwarcfiter. Finding Elementary Cycles of a Directed Graph in $O(n + m)$ per Cycle. *No.* 60*, University of Newcastle upon Tyne*, Newcastle upon Tyne, England. 1974.

[30] L. M. Maxwell, G. B. Reed. Subgraph Identification-Segs, Circuits and Paths. 8*th Midwest Symposium on Circuit Theory*, Colorado State University, pp. 13.0-13.10. 1965.

[31] R. McConnell. A Certifying Algorithm for the Consecutive Ones Property. *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 768-777. 2004.

[32] J. Meidanis, O. Porto, G. P. Telles. On the Consecutive Ones Property. *Discrete Applied Mathematics*, 88, pp. 325-354. 1998.

[33] J. Meidanis, J. C. Setubal. Physical Mapping of DNA. *Introduction to Computational Molecular Biology*, pp. 143-173. 1997.

[34] V. G. K. Murti, V. V. K. Rao. Enumeration of All Circuits of a Graph. *Proceedings of the IEEE*, 57, pp. 700-701. 1969.

[35] J. Ponstein. Self-avoiding Paths and Adjacency Matrix of a Graph. *SIAM Journal on Applied Mathematics*, 14, pp. 600-609. 1966.

[36] R. C. Read, R. E. Tarjan. Bounds on Backtracking Algorithms for Listing Cycles, Paths, and Spanning Trees. *Networks*, 5, pp. 237-252. 1975.

[37] J. Stoye, R. Wittler. A Unified Approach for Reconstructing Ancient Gene Clusters. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 99. 2008.

[38] J. Tan, L. Zhang. Approximation Algorithms for the Consecutive ones Sub-matrix Problem on Sparse Matrices. *Proceedings of ISAAC*, LNCS, 3341, pp. 835-846. 2004.

[39] R. E. Tarjan. Depth-first Search and Linear Graph Algorithms. *IRE Transactions*, 1, pp. 146-160. 1972.

[40] J. C. Tiernan. An Efficient Search Algorithm to Find the Elementary Cycles of a Graph. *Communications of the ACM*, 13, pp. 722-726. 1970.

[41] A. Tucker. A Structure Theorem for the Consecutive 1s Property. *Journal of Combinatorial Theory (B)*, 12, pp. 153-162. 1972.

[42] L.G.Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM Journal of Computing* , 8, pp. 410-421. 1979.

[43] J. T. Welch Cycle Algorithms for Undirected Linear Graphs and Some Immediate Applications. *Proceedings of the ACM National Conference*, pp. 296-301. 1965.

[44] M. Wild. Generating All Cycles, Chordless Cycles, and Hamiltonian Cycles with the Principle of Exclusion *Journal of Discrete Algorithms*, 6, pp. 93-102. 2008.

[45] S. S. Yau. Generations of All Hamilton Circuits, Paths, and Centers of a Graph, and Related Problems. *IEEE Transactions on Circuit Theory*, 14, pp. 79-81. 1967.